REMOTE MONITORING OF RAILWAY EQUIPMENT USING INTERNET TECHNOLOGIES

TXOMIN NIEVA [a, b], ANDREAS FABRI [b], ALAIN WEGMANN [a]

[a] Institute for comp. Communications and Applications (ICA)
Communication Systems Dept. (DSC)
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
http://icawww.epfl.ch
{txomin.nieva, alain.wegmann}@epfl.ch

[b] Industrial Software Systems CHCRC.C2
Information Technologies Dept.
ABB Corporate Research Ltd.
CH-5405 Baden, Switzerland
http://www.abb.ch/chcrc
{txomin.nieva, andreas.fabri}@ch.abb.com

ABSTRACT

This paper outlines the main benefits of using Internet technologies for the remote monitoring of railway equipment. We present two prototypes of a remote monitoring tool for railway equipment. The first has a 2-tier architecture and is based on Java technology and Java RMI as a communication protocol. The second has a 3-tier architecture and is based on XML/XSL technology and HTTP as a communication protocol. We compare both systems and we give some conclusions from the actual work. This paper is intended for people concerned with industrial applications on the Internet and especially for those developing remote monitoring tools for embedded systems.

KEYWORDS

Internet Computing, XML/XSL, Java, Remote Monitoring, Railway Equipment

1. INTRODUCTION

The Internet, having caused a revolutionary impact on office automation, is currently heavily influencing the industrial automation and information systems. The emergence of the Internet provides a framework for communication with any piece of hardware and software, independently of where it is physically located. Heterogeneous distributed embedded systems, which were commonly isolated in the past, are increasingly connected to networks and integrated within information systems. The management of distributed embedded systems is becoming an immense task for embedded systems providers, operators, and service organizations that want to offer their customers a high quality of service. The interconnection of distributed embedded systems and information systems brings significant benefits and offers new business opportunities. One of the applications of the Internet in the industry is for the remote monitoring of embedded systems. Some examples of these applications can be seen at [1-6].

This paper is organized as follows: First, we describe the RoMain system, a remote monitoring tool for railway equipment, outlining the main benefits of using the Internet and Internet technologies for the remote monitoring of railway equipment. Second, we describe a prototype of this system based on Java technology and Java communication middleware. Third, we describe another prototype of the same system based on XML/XSL technology and HTTP as communication protocol. Then, we compare both prototypes, mainly regarding the communication performances. Finally, we give some conclusions from the actual work.

2. THE ROMAIN SYSTEM

We developed, in the frame of the *Railway Open System Interconnection Network (ROSIN)* European project, a web-based monitoring tool for trains that supports maintenance work. This monitoring tool was called Railway Open Maintenance tool (RoMain) [1, 7]. The objective of this tool was not to replace the existing control network, but rather to enhance it with a parallel low-cost on-line data network for railways, in order to support maintenance work. This data network will allow maintenance staff to supervise railway equipment from anywhere at anytime. It will also enable experts at different locations to collaborate and to anticipate maintenance tasks. The user requirements for such a tool were: (i) ubiquitous access, (ii) low cost, (iii) user friendly interface with textual and graphical views of the information and (iv) easy update of equipment documentation. Taking into account all these requirements, we decided to take an approach based on the Internet. The Internet has had a revolutionary impact on office automation, and now there is a clear trend towards using Internet technologies for industrial automation. The introduction of Internet technologies for accessing embedded systems is mostly cost driven, thus bringing significant benefits:

(i) Reduction of the development costs of an application, by enabling the use of Common Off-The-Shelf (COTS) software components.

(ii) Elimination of the costs of a proprietary communication network, by using the common Internet network.
(iii) Reduction of the costs of development of a client application for each different platform, by using a standard web-browser as a single client interface for heterogeneous platforms.
(iv) Elimination of the costs of installing proprietary client applications, as the client interface is a standard web browser usually pre-installed on the client machine.
(v) Reduction of the costs of maintaining up-to-date equipment documentation, by offering a simple way (hyperlinks) to publish documents accessible immediately from anywhere in the world.
(vi) Reduction of maintenance personal travel costs, by the possibility of ubiquitous access to the information.
(vii) Reduction of maintenance scheduling costs, by the possibility of ubiquitous access to the information at any time.

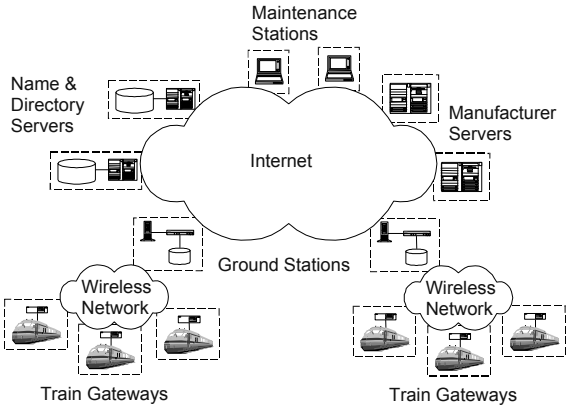


Figure 1. The RoMain System

The architecture of the RoMain system, shown in Figure 1, is composed of:

(i) *Train Gateways* - connected to the train network gather actual train data.
(ii) *Ground Stations* - automatically establish connections to train gateways over wireless networks.
(iii) *Name and Directory Servers* - provide information about the train component models and train directory.
(iv) *Manufacturer Servers* - provide on-line information about train components, for example fact sheets, user manuals, or installation instructions.
(v) *Maintenance Stations* - run a standard web browser to access train data.

All these systems are interconnected by means of a secure TCP/IP network, usually the Internet, or eventually an Intranet or Virtual Private Network.

In the following sections, we describe the different prototypes that we developed of the RoMain system.

3. ROMAIN JAVA: MONITORING OF ALL DEVICES ON A SINGLE TRAIN

The first prototype has a 2-tier architecture and is based on Java technology and Java Remote Method Invocation (Java RMI) [8-10] as communication protocol. Java, promoted by Sun Microsystems, is basically a programming language and a running platform. The Java language is an easy to learn but quite efficient object-oriented programming language; the Java Virtual Machine (JVM) enables platform independence; and the Java Application Programming Interface (API) provides software developers with a rich library of classes. The Java API provides many ways to enable network connectivity. One of them is Java RMI, which is a communication middleware that enables communication between objects running in different locations. Java RMI brings a Remote Procedure Call (RPC) like mechanism to execute methods of object located remotely.

The main goal of this prototype was to specify an API for a Data Acquisition System (DAS) on-board a train. This API defines the interface between client and server, and it therefore allows for the implementation of new client applications. As the API is object-oriented and as the applications are distributed, we had a choice between different middleware products: DCOM [11, 12], CORBA [11, 13, 14], or Java RMI. We opted for the latter as we strived for a 100% pure Java solution. Therefore, we developed the on-board DAS as a Java RMI server. It offers remote interfaces for, discovering train configuration, and accessing train, vehicle and equipment data - to give just two examples. Based on the API we developed a client system, as a Java RMI client within a Java applet, that uses these remote interfaces to display the current state of a train within a web browser.

Two different updating mechanisms were implemented based on pull and push technologies. Using push technology the update of data is triggered by the server, if and only if, there are changes in that data. Hence a GSM connection is only open during notification subscription and notification, even if there is an arbitrary, long time span in between.

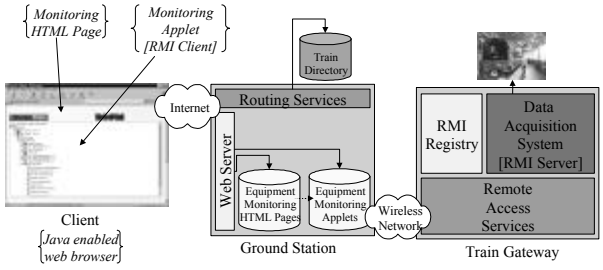


Figure 2. Monitoring of all Devices on a Single Train

The architecture of the system is shown in Figure 2. The remote train gateway is usually reachable by means of a wireless network, in our case by GSM. A ground station is responsible for transparently establishing a TCP/IP connection to the remote train gateway via this wireless network. As the bandwidth of GSM is still relatively low, the downloading of the monitoring HTML page, plus the downloading of the associated monitoring Java applet, is slow. In order to improve the performance of this download, we investigated an alternative: The monitoring HTML page and the Java applet were moved from the remote train gateway to a ground based web server, in our case to the ground station. The train gateway then became a pure data server.

In this prototype we also investigated security issues. As the user must grant the client Java applet certain privileges for stepping out of the sandbox (the restrictive security policy implemented by the browser), he must be able to check whether this Java applet is trustworthy or not. Therefore, the developer of the Java applet must sign it with a digital signature obtained from a certificate authority. We use the same technology that is used to make e-commerce applications more secure.

The main problem with this prototype is the problem of accessing data for a client behind a firewall. This can be partially overcome with HTTP tunneling, but this slows down the communication and the use of server side push technology is no longer possible.

A demonstration of this prototype can be found at http://icapc62.epfl.ch/romainjava/. In this demonstration, a train gateway is connected to a train network installed in a laboratory at the *CAF* (a Spanish train manufacturer) facilities in *Beasain (Spain)*. This train network is exactly the same as the network that was installed on a real train for a demonstration in February'99. The data is collected from real devices, which are interconnected via the Train Communication Network (TCN) [15]. The ground station is installed at the ICA institute. The only difference with the real demonstration is that the connection of the ground station with the train gateway is not done by a wireless network but by an Internet wired connection. This is done in order to reduce costs. However, this would be totally transparent for any client using the application.

4. ROMAIN XML: MONITORING OF ALL DEVICES ON A FLEET OF TRAINS

The second prototype has a 3-tier architecture and is based on XML/XSL [12, 16-18] technology and HTTP as communication protocol. The eXtensible Markup Language (XML) is the "de facto" standard for data exchange over the Internet. This new standard was specified, similarly to the Hypertext Markup Language (HTML), by the *World Wide Web Consortium (W3C)* from a subset of the historical Standard Generalized Markup Language (SGML). XML data looks very much like HTML. However, XML allows developers to define a specific grammar for a specific application. This grammar can be specified by the means of a Document Type Definition (DTD). There are already many standard DTDs that were agreed upon by companies working in the same business domain. These standards enable data exchange among heterogeneous systems. XML data is easy to create, parse, combine and transform into other formats. The eXtensible Style Language (XSL) is an advanced style sheet language designed for the use with well-formed XML documents. XSL documents contain a series of XSL elements that apply to particular patterns of XML documents. When a particular XML pattern is found, the XSL element outputs a combination of text. The HyperText Transfer Protocol (HTTP) is an application protocol that defines a set of rules for exchanging data over the Internet. HTTP is based on TCP/IP, the transport and session standard protocols of the Internet.

In this prototype we implemented a system with a three-tier architecture to investigate how data from an entire fleet of trains can be integrated, but also to overcome the problem we had with the previous prototype. This allows a component manufacturer to supervise, for example, all door controllers, regardless on which trains they are. We investigated technologies that allow the client to choose among different views, and to receive data combined from train gateways on a single page. Choosing among different views means that we need a way to separate what is data and what is presentation format. Combining data from different sources means that we have to parse the data and create new data. All these features are easily implemented by the use of XML. As XML also enables the defining of new domain specific markup languages, data can be transmitted together with some metadata that describe them. This makes it easy to parse and combine XML documents. Moreover, the presentation format can be added by means of a separate XSL file, which defines how an XML document should be displayed. Therefore, it is easy to implement different views of the same XML document by providing different XSL files.

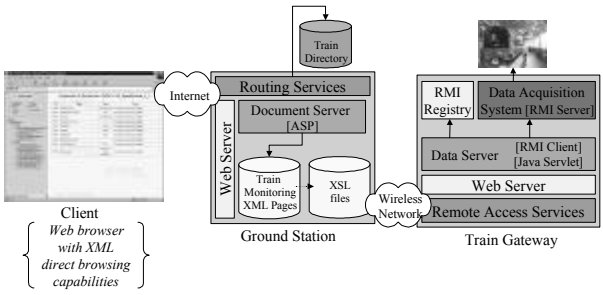


Figure 3. Monitoring of all Devices on a Fleet of Trains

The architecture of the system is shown in Figure 3. It has a three tier architecture composed of: (i) a Java servlet

[19] on-board a train that gathers data from the Java RMI server developed for the second prototype and that replies to an HTTP request of data with XML documents giving the requested information; (ii) a middle tier that receives data from different sources in XML format, combines them into a single XML document and adds the style sheet corresponding to the client view; (iii) and the thin client, which is like in the previous prototype, a web browser. Note that in the first prototype the Java applet also is loaded from the ground station, but then it connects directly to the data server on the train; this means in the first prototype we had a two-tier architecture.

As in the previous prototype, a ground station is used in order to access, via a wireless network, the remote train gateway. In this prototype there is no monitoring HTML page or Java applet. Instead there is a document server on the middle-tier, which is responsible for integrating data from different train gateways and for adding the corresponding style sheet to make the output readable by the client. In our case, the entire middle-tier is on the ground station.

This architecture is scalable, extensible and adaptable for future evolution, and it runs over firewalls. The main drawback is that it is not possible to use server side push technology.

A demonstration of this prototype can be found at http://icapc62.epfl.ch/romainxml/. In this demonstration, the train data source is the same as in the demonstration of the previous prototype. The ground station is installed at *ICA*. The communication with the train gateway is done by means of an Internet wired connection.

5. ROMAIN JAVA VS. ROMAIN XML

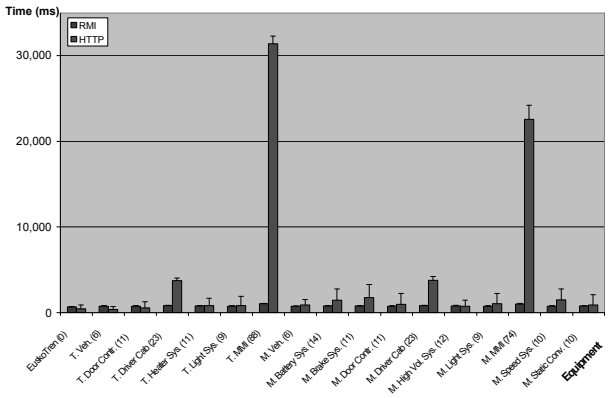
In this section we present and discuss a comparison of the RoMain Java prototype versus the RoMain XML prototype, under the following criteria: communication performance, client interface, scalability, security, reliability, availability and evolvability.

5.1. COMMUNICATION PERFORMANCE

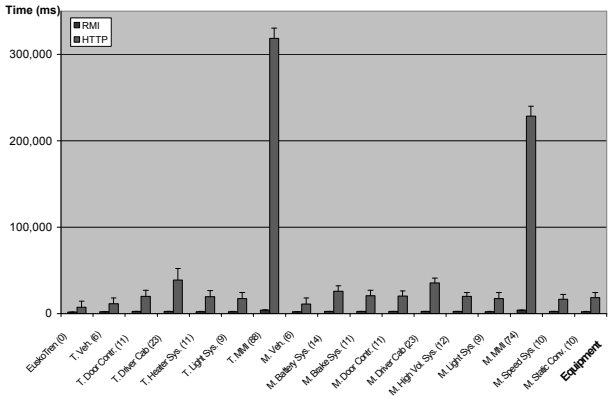
In both prototypes, in order to compare the communication performance, we measured the response time to obtain, for each equipment, the current values of all the properties. A property of an equipment is an attribute that gives some information about it and its current state. In both prototypes the data source is a train network installed in a laboratory at the *CAF* facilities. The ground station and the clients are installed in the same machine at *ICA*. This is because we wanted to compare the communication performance of Java RMI versus HTTP in a single client-server communication. The overhead of the three-tier communication of the RoMain XML prototype is not taken into account. The real Internet is used as communication network. We used Java 1.1.7 to develop the Java RMI server and the performance clients for this evaluation. These clients use the pull mechanism to obtain the current values of the properties.

To obtain more reliable results we did 100 essays of each measurement. We represent the average and standard deviation of the results into a chart diagram. The X-axis represents the different equipment, with the number of properties in brakes. The Y-axis represents the time in milliseconds. A column bar represents the average time in milliseconds to obtain all the properties of an equipment. A line appended to a column represents the standard deviation of the measurements.

The first evaluation consisted in measuring the time to obtain, for each equipment, 1 update of all its properties. The results of this evaluation are shown in Figure 4a.



(a) 1 Update

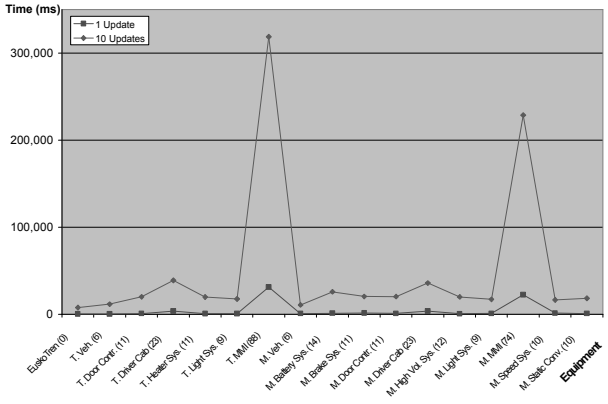


(b) 10 Updates

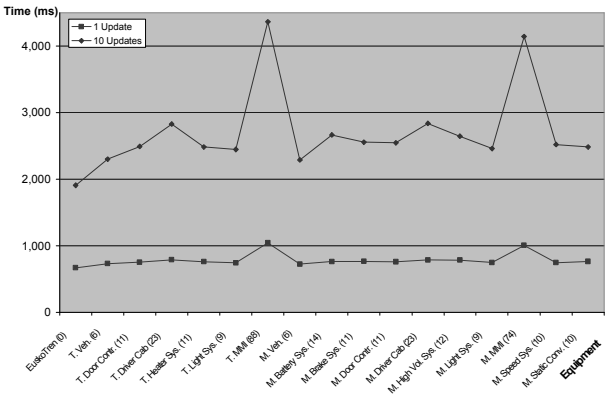
Figure 4. RoMain Java vs. RoMain XML Communication Performance Comparison

These results demonstrated that, with 1 update of the properties, the performance of Java RMI is generally much better than the performance of HTTP. Only in the case of an equipment with few properties (that is the case of "Euskotren") the performance of HTTP is slightly

better than the performance of Java RMI. The more properties an equipment has, the longer the difference is between the performances of Java RMI against HTTP. In the case of an equipment with a huge quantity of properties (this is the case of "T.MMI" and "M.MMI") the performance of Java RMI is substantially higher than the performance of HTTP. Therefore, we can conclude that the RoMain Java prototype demonstrated a better communication performance than the RoMain XML prototype.



(a) RoMain XML



(b) RoMain Java

Figure 5. 1 Update vs. 10 Updates Comparison

In the RoMain XML prototype, a client application has to perform a new request to update the properties of an equipment. Therefore, we presume that the cost of a second and subsequent updates will be the same as the cost of the first update. However, in the RoMain Java prototype, in the first update a client application initially obtains a local reference (a proxy object) to the remote object, and then it invokes a remote call to this object to obtain all the properties of an equipment. In successive updates of the properties, it no longer needs to obtain the proxy object. It has only to invoke a remote call to obtain all the properties again. Thus, in Java RMI the cost of a second and subsequent updates should be lower than the cost of the first update. In order to corroborate this we performed a second evaluation. This evaluation consisted in measuring the time to obtain, for each equipment, 10 updates of all its properties. The results of this evaluation are shown in Figure 4b.

These results demonstrated that with 10 updates of the properties, the performance of Java RMI is always much better than the performance of HTTP. Even more, the differences between the performance of Java RMI and HTTP have increased substantially. This is because the time to obtain 10 updates of all properties in the HTTP based prototype is, on average, about 10 times longer than the time to obtain the first update. This comparison is shown in Figure 5a. However, in the Java RMI based prototype, the time to obtain 10 updates of all properties is, on average, less than 4 times longer than the time to obtain the first update. This comparison is shown in Figure 5b. The reason for these differences between Java RMI and HTTP is that in Java RMI the cost of successive updates is lower than the cost of the first update. However, in HTTP the time to obtain "n" updates corresponds approximately to "n" times the time to obtain the first update.

Therefore, we can state that the difference between the performances of Java RMI against HTTP will increase as we increase the numbers of updates. In conclusion, Java RMI will perform even better than HTTP when numerous updates are required.

5.2. CLIENT INTERFACE

In both prototypes, clients use Internet browsers to access the current state of the properties of a train. But the mechanisms to present this data to clients are very different.

In the RoMain Java prototype, the client interface is implemented as a Java applet. Java applets run on many standard browsers (such as Netscape Communicator and Microsoft Internet Explorer) without any pre-installation. We made use of a graphical library, called "Swing", to present graphically the current state of the properties of a train. This library is an extension to the standard Java API, and it may not be locally installed with a specific version of a browser. In this case, a client would need to pre-install locally this library before running the Java applet. Java applets and Swing offer a very powerful set of graphical components to design graphical interfaces. In order to save wireless communication bandwidth, the Java applet is downloaded from a ground station and not from the train gateway. Once the Java applet is running on the client browser, the client-server communication is established directly between the client Java applet and the train gateway. A security restriction of Java applets in Java 1.1 allows a Java applet to establish communication only with the server it was downloaded from. To overcome this problem we used special classes that allow

us to give special privileges (such as the communication with any server) to Java applets. The problem is that the classes we used are specific to the Netscape Communicator browser. Hence, the RoMain Java prototype only works using this browser. Similar mechanisms exist for Microsoft Internet Explorer but there is no cross browser compatibility. The problem of the security restriction of Java applets is currently solved with Java 1.2. This new version of Java provides a policy-based, easily configurable, fine-grained access control.

In the RoMain XML prototype, we use XML direct browsing to generate on the client side the view with an XSL file. XSL has demonstrated to be a powerful means to combine and transform XML files into another XML file or another presentation format (such as HTML). XML/XSL direct browsing capabilities offer an elegant manner to generate an XML presentation directly on the client side. Unfortunately, direct browsing of XML files is currently only possible by using *Microsoft IE5* as an Internet browser. A possible solution to enable clients with browsers without XML direct browsing capabilities is to transform, on the server side, the XML data into plain HTML. In this way, the data could be displayed on any standard browser.

Using Java applets instead of XML direct browsing for the client view brings significant benefits. A Java applet is an application running on a web browser. Therefore, a Java applet can make possible the implementation of complex features such as receiving notifications of server side updates of data. However, the cost of downloading a Java applet is always significantly higher than the cost of direct browsing of XML. Additionally, direct browsing of XML allows for an efficient switching between different client views at the client side, without performing a new communication. An important advantage of Java applets is that they can be displayed on many browsers, while direct browsing of XML is only possible, today, using Internet Explorer 5.

5.3. SCALABILITY

In both prototypes the bottleneck is obviously the train gateway, which is implemented as a Java RMI server. Java RMI servers can support efficiently a few hundred simultaneous clients, but they are not scalable for large systems. Additionally, the performance slows down as the number of clients increase. In the case of our application - the remote monitoring of railway equipment - we do not expect a large amount of simultaneous clients. Therefore, the limitation on the scalability of Java RMI servers does not cause any real problems.

In the RoMain Java prototype, the communication is established directly between client Java applets and the Java RMI server. If scalability had been an important requirement of our system, we would have designed the application in a different way. One solution to designing scalable systems using Java RMI is to implement a middle tier with many application servers, each of them handling requests from many client Java applets. Each application server establishes a single Java RMI communication with the Java RMI server on the train gateway. A load-balancing manager would be responsible to assign, at runtime, an application server with enough resources to a client Java applet. In Java RMI does not exists a pre-defined load balancing manager, but it should not be too much complicated to develop such a system.

In the RoMain XML prototype, it is easier to scale the system because it is implemented in a three-tier architecture. In the middle tier an application server handles requests from Internet clients and dispatches the requests to a data server installed on the same machine as the Java RMI server. If we need a very scalable system, we have only to deploy the data server on many machines. Then, a load-balancing manager on the application server would decide, at run-time, which data server to contact to dispatch the request to the Java RMI server on the train gateway.

5.4. SECURITY

In both prototypes standard mechanisms for authentication of users can be easily implemented. In the RoMain Java prototype, the authentication of the user can be done by a Java applet that is downloaded from the ground station before downloading the monitoring Java applets. Once a user has been authenticated, the user is allowed to download the monitoring Java applets. In the RoMain XML prototype, the authentication mechanisms can be implemented in the ground station as well, as part of the application server.

Both Java RMI and HTTP support the Secure Socket Layer (SSL) for encryption of the information before transmitting it over the Internet. SSL provides simple but efficient mechanisms to encrypt the information in ways that renders hacking difficult. Despite other more complex and efficient mechanisms for security, SSL has become the de-facto standard over the Internet for encryption of data.

5.5. RELIABILITY

Both prototypes depend heavily on the network. Therefore, the reliability of such systems depends very much on the reliability of the network itself. In our experiment we used the Internet as a communication network. Due to the nature of the Internet and its fluctuant bandwidth, this network cannot be considered reliable.

Analyzing the results of the evaluations we noticed that the differences between the same measurements in the RoMain XML prototype are larger than the differences between the same measurements in the RoMain Java prototype. This is due to the fact that the RoMain XML prototype requires more use of the network

than the RoMain Java prototype, to obtain the same information. Additionally, as the RoMain XML prototype is implemented in a three-tier architecture it makes even more use of the network than the RoMain Java prototype. Effectively, two HTTP calls are needed to obtain the current values of the properties of an equipment and send these values back to a client application. However, in the RoMain Java prototype an update of the properties of an equipment needs only a Java RMI remote call.

In conclusion, we can state that the RoMain Java prototype is more reliable than the RoMain XML prototype.

5.6. AVAILABILITY

The RoMain XML prototype proposes a three-tier architecture, whereas the RoMain Java prototype proposes a two-tier architecture. Therefore, in the RoMain XML prototype there are more agents that interact within the system than in the RoMain Java prototype. Effectively, in the RoMain XML prototype we have a data server that obtains data from a DAS in the train gateway, a document server in the ground station and a client running IE5.

In the RoMain Java prototype, there is only one Java RMI server that obtains data from a DAS in the train gateway and the Java RMI client running within a Java applet in the client's browser. The probability of failure in the RoMain XML prototype is increased by the inclusion of a middle tier.

In conclusion, the RoMain Java prototype has a higher availability than the RoMain XML prototype.

5.7. EVOLVABILITY

This is the point that makes the RoMain XML prototype really interesting. The RoMain XML prototype is based on XML. XML is easy to create, parse and transform. It is very simple to integrate data coming from different sources, using XSL style sheets. In the RoMain XML prototype the integration of data coming from different trains is easily done at the middle tier. The RoMain XML prototype offers an architecture that allows a high flexibility in evolution.

The RoMain Java prototype provides a good example of client-server computing. However, it is more complicated to integrate data coming from different trains and to process the data of a train in different ways as is done in the current prototype - just to give some examples.

In conclusion, the RoMain XML prototype brings a higher evolvability than the RoMain Java prototype. A solution to increase the evolvability of the RoMain Java prototype is to re-design this system with a three-tier architecture. As in the RoMain XML prototype, a middle tier would be responsible for establishing connections and retrieving data from different trains. Furthermore, it would be responsible for integrating this data and sending it back to a client.

6. CONCLUSION

The RoMain Java prototype demonstrated a better communication performance than the RoMain XML prototype. The prototype based on Java RMI has the additional advantage that it can use server side push technology. This can optimize data communication by only transmitting data that has been changed on the monitored system. The main drawback of this prototype is that clients within firewalls cannot efficiently use it. It is possible to use Java RMI over firewalls by using the concept of HTTP tunneling, but the efficiency of the communication slows down considerably. The HTTP based prototype enables clients to use the prototype even within firewalls. However, the communication performance is low. Eventually, this performance can be improved by compressing the XML formatted data (using an algorithm as ZIP) on the server and by uncompressing it again on a client. The client interface offered by the RoMain Java prototype allows for the implementation of complex features such as receiving notifications of server side updates of data. The client interface of the RoMain XML prototype is much more simple but it allows for switching between different client views at the client side. The scalability and reliability of both prototypes are acceptable for systems like RoMain. The reliability and availability of the RoMain Java prototype are higher than in the RoMain XML prototype. This is mainly due to the fact that the RoMain XML prototype proposes a three-tier architecture, whereas the RoMain Java prototype proposes a, much simpler, 2-tier architecture. For the same reasons, the RoMain XML prototype is also much more flexible in evolution than the RoMain Java prototype.

The conclusion of these experiments is that when a high performance remote monitoring system is required, Java and Java RMI are the right technologies. If flexibility on evolution is a strong requirement, a three-tier system, which is rather simple to develop using technologies such as HTTP and XML, may be a better choice.

ACKNOWLEDGMENTS

Thanks to all the members of the *ROSIN WP4,* who brought a real framework to the discussions and the implementation of our hypotheses. We want to thank especially to *Hubert Kirrmann* from *ABB Corporate Research*, for his many contributions to this work, and to *Xabier Arizkorreta* from *CAF,* for his invaluable support during the performance measurements. Thanks also to *Jan Ellerbrock*, who did excellent work implementing the RoMain XML prototype as part of his diploma thesis.

REFERENCES

[1] A. Fabri, T. Nieva, and P. Umiliacchi, Use of the Internet for Remote Train Monitoring and Control: the ROSIN Project, *Rail Technology '99*, London, UK, September 7-8, 1999, http://icawww.epfl.ch/nieva/thesis/Conferences/RailTech99/article/RailTech99.PDF.

[2] M. P. de Albuquerque and E. Lelievre-Berna, Remote Monitoring over the Internet, *Nuclear Instruments & Methods in Physics Research*, 412(1), 1998, 140-145.

[3] F. Olken, H. A. Jacobsen, C. McParland, M. A. Piette, and M. F. Anderson, Objects lessons learned from a distributed system for remote building monitoring and operation, *Conference on Object-oriented Programming, Systems, Languages and Applications*, Vancouver, Canada, October 18-22, 1998, http://www.lbl.gov/~olken/rbo/rbo.html.

[4] K. Kusunoki, I. Imai, H. Ohtani, T. Nakakawaji, M. Ohshima, and K. Ushijima, A CORBA-based Remote Monitoring System for Factory Automation, *First International Symposium on Object-Oriented Real-time Distributed Computing - ISORC'98*, Kyoto, Japan, April 20-22, 1998.

[5] R. Itschner, C. Pommerell, and M. Rutishauser, *GLASS: Remote Monitoring of Embedded Systems in Power Engineering,* IEEE Internet Computing, 2, 1998, 46-52.

[6] T. Lumpp, G. Gruhler, and W. Küchlin, Virtual Java Devices: Integration of Fieldbus Based Systems in the Internet, *Annual Conference of the IEEE Industrial Electronics Society - IECON'98*, Aachen - Germany, August 31 - September 4, 1998.

[7] T. Nieva, Automatic Configuration for Remote Diagnosis and Monitoring of Railway Equipment, *IASTED International Conference - Applied Informatics*, Innsbruck, Austria, February 15-18, 1999, http://icawww.epfl.ch/nieva/thesis/Conferences/ai99/article/ai99.pdf.

[8] Sun Microsystems, *The Java Tutorial*, February, 2000, http://java.sun.com/docs/books/tutorial/index.html.

[9] D. Flanagan, *Java in a Nutshell*, (Paris: O'Reilly & Associates Inc., 1996).

[10] Sun Microsystems, *Java Remote Method Invocation White Paper*, 1999, http://java.sun.com/marketing/collateral/javarmi.html.

[11] R. Orfali, D. Harkey, and J. Edwards, *The Essential Client/Server Survival Guide*, 2nd ed, (New York: John Wiley & Sons Inc., 1996).

[12] Microsoft, *Microsoft Developer Network (MSDN) Online*, 2000, http://msdn.microsoft.com/.

[13] A. Vogel and K. Duddy, *Java Programming with CORBA*, (New York: John Wiley & Sons Inc., 1997).

[14] OMG, *OMG Home Page*, 2000, http://www.omg.org/.

[15] IEC, "*Electric Railway Equipment - Train Bus - Part 1: Train Communication Network*", IEC 61375-1, 1999.

[16] W3C, *eXtensible Markup Language (XML) Home Page*, 1997, http://www.w3.org/XML/.

[17] C. F. Goldfarb and P. Prescod, *The XML Handbook*, (Upper Saddle River, NJ: Prentice Hall Inc., 1998).

[18] N. Bradley, *The XML Companion*, (Essex: Addison-Wesley Longman Limited, 1998).

[19] J. Hunter and W. Crawford, *Java Servlet Programming*, (Sebastopol: O'Reilly & Associates Inc., 1998).