WWW-BASED REMOTE CONTROL USING TINI CARDS AND *BRAZIL*

J.J. Elosua, J.C. Burguillo

Departamento de Ingenieria Telemática.
ETSI Telecomunicación, Universidade de Vigo.
E-36200 Vigo (Pontevedra), SPAIN
Phone/Fax: +34-986 813869 / +34-986 812116.
E-mails: jjelosua@mundo-r.com, jrial@ait.uvigo.es

Abstract: In this paper a fast, compact, cost-effective solution to build Web-based remote control systems is presented. This solution has been developed using a compact architecture given by the powerful combination of TINI cards (Tiny InterNet Interface) and the *Brazil* platform. This infrastructure has been combined and put into practice building a system that controls a robot and a video camera. This system is entirely Java-based so cross-platform, security and network facilities are inherited.

Keywords: Remote control, Mobile Robot, Telematics, Internet, Java.

1. INTRODUCTION

During the last years we have seen an unstoppable expansion of the Internet. Having this in mind, in the world of control engineering it is an added value to provide Web-based interfaces to many types of devices. Nowadays many devices do not have this interface implemented and even the suitability of making this interface internal could be discussed in terms of cost, security and efficiency. Hence, it is very interesting to have a small, low-cost, external component to increase the functionality for a wide range of control devices.

The TINI card (Loomis, 2001) matches perfectly the previous need, this small card was designed to give a voice on the network to many types of devices: from small sensors and actuators to factory automation equipment and legacy hardware. TINI has a built-in Java run time environment. Java (Sun, n.d.*d*) was not commonly used for embedded systems, but today the relative cost of programming rises compared with hardware costs, so better use of programmers is needed. This situation fits perfectly with the Java philosophy (Roussel and Duris, 2000).

The *Brazil* project (Sun, n.d.*a*) developed by Sun Labs started off as an HTTP stack designed with a very small footprint ($< 100KB$). It evolved into a more general toolkit for putting URL-based interfaces on a wide range of applications and devices. *Brazil* has been chosen for the development of the system because it gives an homogeneous structure to any complex system that includes small devices, traditional Web applications and big meta-servers.

Nowadays WWW-based applications for remote control systems are emerging in a variety of fields such as tele-education (Rodriguez *et al.*, 2001), telerobotics (Behnke and Elzer, 2001), home automation (Skrtic *et al.*, 2001) and industrial environments (Schmid *et al.*, 2001).

The architecture presented in this paper presents several advantages compared with traditional ones:

- *micro-server*: Made from the combination of the TINI card and *Brazil*, instead of the traditional PC or workstation to serve the requests. This *micro-server* offers a gain in mobility, space and cost.

- *UPI & Java*: The *Brazil* technology is based on the URL programming interface (UPI). The UPI interface provides a set of URLs to any device available through the HTTP protocol. The highest degree of openness is therefore achieved because HTTP protocol is supported by any commercial Web browser. Beyond that, Java provides the required security to make this openness truly reliable.
- *code reuse*: *Brazil* functionality is built from very small software pieces. Therefore, it is really easy to reuse or reorganize those pieces to build new applications. An example illustrating how to share those pieces on the Web may be found in (DiGiorgio, n.d.).

The rest of this paper is organized as follows. Section 2 describes the main features provided by TINI cards. Section 3 introduces the *Brazil* platform. Section 4 describes a feasible combination of this infrastructure to develop and construct the Web-based remote control system. Finally, in section 5, conclusions and further work are discussed.

2. TINY INTERNET INTERFACE: TINI

The TINI platform is a combination of a small but powerful chip-set and a Java programmable runtime environment. The chip-set provides processing, control, device-level communication and networking capabilities. The features of the underlying hardware are exposed to the software developer through a set of Java API's (Application Programming Interfaces).

Also, the combination of broad-based I/O capability, a TCP/IP network protocol stack, and a Java programming environment empowers programmers to quickly create applications that provide not only local control but also global access to TINI-based devices. This connectivity of this TINI-based devices is therefore extended by allowing interaction with remote systems and users through standard network applications such as Web browsers.

2.1 *TINI Hardware*

Although TINI does not require a specific hardware design, a TINI board model 390 (TBM390) is available. TBM390 is a compact (31.8mm X 102.9 mm) 72-pin SIMM board that allows both hardware and software designers to begin prototyping and development work without a large up-front investment of either money or time. This board model provides the following important features:

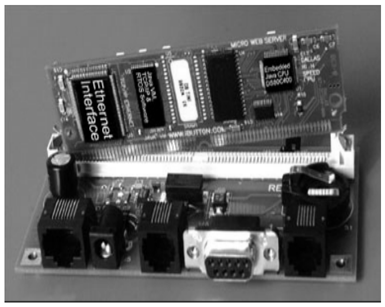
Fig. 1. The TINI card

- Currently the DS80C390 microcontroller: is the heart of any TINI hardware design and directly executes the native code portion of the runtime environment.
- 512KB Flash ROM: stores TINI's runtime environment.
- 512KB persistent SRAM(expandable to 1MB): contains the system data as well as the garbage collected heap from which all Java objects are allocated. It also stores all file system data.
- 10Base-T Ethernet controller and Real-time clock.
- Dual CAN[1] controllers, Dual 1-Wire net interface, Dual serial port and 2-wire synchronous serial port
- Exposes the microcontroller's address and data buses for parallel I/O expansion and requires only a single +5V power supply.

There is a need for a socket board to provide physical connectors to interface the TBM390 with other equipment such as a Ethernet network, a serial device, or a 1-Wire network.

2.2 *TINI runtime environment*

Providing hardware for developing embedded network devices is only half the job. A large amount of software is also required to free application developers from having to worry about the details of creating layers of infrastructure. For this reason a runtime environment was developed from the beginning as an integral part of the overall platform.

The software that comprises TINI runtime environment can be divided into two categories: native code executed directly by the microcontroller and an API interpreted as bytecodes by the Java Virtual Machine.

2.3 *TINI applications*

Here are a few of the possible applications that can be build using this technology:

[1] CAN (Controller Area Netwotk) fieldbus is widely used in industrial automation systems

- *Industrial Controls*: TINI's integrated CAN support is instrumental in implementing factory automation equipment, network switches and actuators.
- *Web-based equipment monitoring and control*: it can be used for communication with equipment to provide remote diagnostics and data collection for purposes such as monitoring device utilization.
- *Protocol Conversion*: TINI-based systems can be used to connect legacy devices to Ethernet networks. Depending on the I/O capabilities of the legacy system, this may also be done by a workstation or a computer. TINI can do the job at a fraction of the cost and size.
- *Environmental monitors*: Using TINI's built-in support for 1-Wire networking, an application can query sensors and report the results to remote hosts.

3. *BRAZIL*

The Brazil project began as an extremely small footprint http stack, originally designed to provide a URL-based interface to smart cards, allowing them to be accessed more easily from an ordinary web browser.

Once the power of this simple Java technology-based code was understood, it evolved into a more general toolkit for putting URL-based interfaces on a wide range of applications and devices.

The Brazil project promotes the functionality of portals and content aggregators. It does this by sitting between the content providers and the users to offer fully personalized and customized content pulled from a variety of independent web sources.

3.1 *Brazil advantages*

This section will try to point out Brazil's strong points:

- *Distributed-Content Web*: Currently, the web consists of browsers talking to web servers. In Brazil's vision, web servers will be augmented by meta-servers, which do the content aggregation and portaling. In addition there will be micro-servers, which produce tiny bits of content that, although not suitable for browsers directly, can be useful in conjunction with the meta-servers. The Brazil framework can be used to build applications for all three levels – web servers, for meta-servers and for micro-servers.

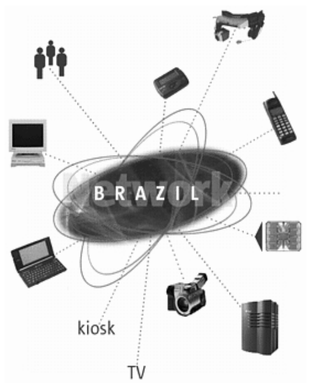

Fig. 2. The Brazil platform

- *Large applications achieved by combining simple parts*: Brazil toolkit's snap-together pieces are called Handlers[2] , The sophistication of your application is really just dependent on how sophisticated you are at putting together these primitive pieces. The Brazil toolkit was designed to make it easy to add new little bits of functionality to an application, combining them with all the bits that are already there.
- *Accelerated Development Cycles*: The idea was to provide a very flexible, malleable toolkit for writing applications very quickly through little interfaces that makes code reuse really easy.
- *More than a Web server*: By using a simple interface, in conjunction with powerful, reusable components, the Brazil technology system is able to deliver a wide range of flexible web solutions, ranging from tiny micro-servers, to traditional web capabilities to fully functional meta-servers that provide sophisticated portal and content aggregation capabilities.

3.2 *Architectural overview*

Typical applications of the Brazil system combine one or more existing components together with custom additions, consisting of one or more implementations of:

- *handlers*: The primary extension interface, which allows for the custom handling of URL's. Just about the only thing they have in common is their lack of dependencies on any other packages. Some provide generic capabilities, such as standard CGI interfaces or template processing, others are either special purpose, provided to demonstrate how to write handlers, and others are skeleton handlers, designed to be *finished* to provide application specific functionality.

[2] A handler is similar to a servlet in Java Web server terminology, but is lighter in weight.

- *filters*: Normally once a handler generates the content for a request, the content is delivered to the client, and the request is finished. Although this allows for a choice in the manner in which the content is obtained, nothing can be done to modify that content before it is transmitted to the client. A filter is a special type of handler, used in conjunction with the FilterHandler that permits content obtained from other handlers to be rewritten or filtered before it is sent to the client.
- *templates*: Are classes that work in conjunction with the TemplateFilter or TemplateHandler that allow HTML/XML content to be processed on a tag by tag basis. The sunlabs.brazil.template.Template interface used to define *templates* does not define the methods used for processing. Instead, the TemplateHandler introspects all of the methods in a template class to determine which ones should be called when HTML tags are seen in the input document. This is an XML like capability that is backward compatible with existing HTML practice.

3.3 *Application development process*

To use *Brazil* server as part of an application, one or more handlers need to be written.

A handler is written by creating a Java class that implements the handler interface. This consists of only two methods, Handler.init(Server, String) and Handler.respond(Request). The first is called once, when the server is initialized, and the other is called upon each HTTP request.

Multiple handlers are needed for most applications, either written specifically for the application or in combination with some of the handlers provided in the sunlabs.brazil.handler package. The Server class utilizes the ChainHandler to allow multiple handlers to work together.

The ChainHandler is the default mechanism used to run multiple handlers. It looks for a single configuration parameter, called handlers, that contains a list of tokens, each of which is the name of another handler. For example, the following entry in a config file:

```
handlers = a b c
a.class = Java class for handler a
b.class = Java class for handler b
c.class = Java class for handler c
```

Call chains of arbitrary depth can be configured this way. The config file handlers property can even refer to other instances of the ChainHandler, enabling the creation of an arbitrary tree of handlers.

4. REMOTE CONTROL SYSTEM

After having looked at the main components of the remote control system's internal skeleton, this section will present, the complete structure and functionality of the system developed.

The aim is to control, using a Web interface, a robot called Khepera (K-team, n.d.), developed by the University of Laussane, and a Sony video camera (EVID30/31) (Sony, n.d.). Figure 3 shows the system structure. In the next paragraphs we detail the system components (hardware and software) and the functionality offered.

4.1 *Hardware components*

During the system development, the next hardware components have been used:

- Khepera Robot: it is the heart of the remote control system and may be controlled through a RS232 serial port.
- Sony Video Camera: Used to track the robot and then send the live video streaming to the remote users so that they could actually follow the system's evolution. This video camera can also be controlled through a RS232 serial port.
- 2 TINI cards: The TINI cards are connected to a Ethernet local network through a hub. Also a RS232-serial connection is made between one TINI and the Khepera, and between the other TINI and the video camera. This is the cornerstone that enables the control of the system through WWW.
- Studio MP10: The Studio MP10 (Pinnacle, n.d.) developed by Pinnacle Systems it was designed for capturing real-time audio and video. This device captures the video streaming produced by the video camera and deploys it into a PC.
- PC: it has two main functions: it acts as a portal for the users of the system, that is, all requests generated by a user will be addressed to this PC. The other function of the PC is to send the live video streaming to remote users using the RTP protocol (Real-Time Protocol) (Ietf, n.d.) designed for the transmission of real-time data.

4.2 *Software requirements*

The following software has been employed in order to develop the application:

- Brazil: The Brazil server runs on both TINI cards and also on the PC acting as the portal for the application.

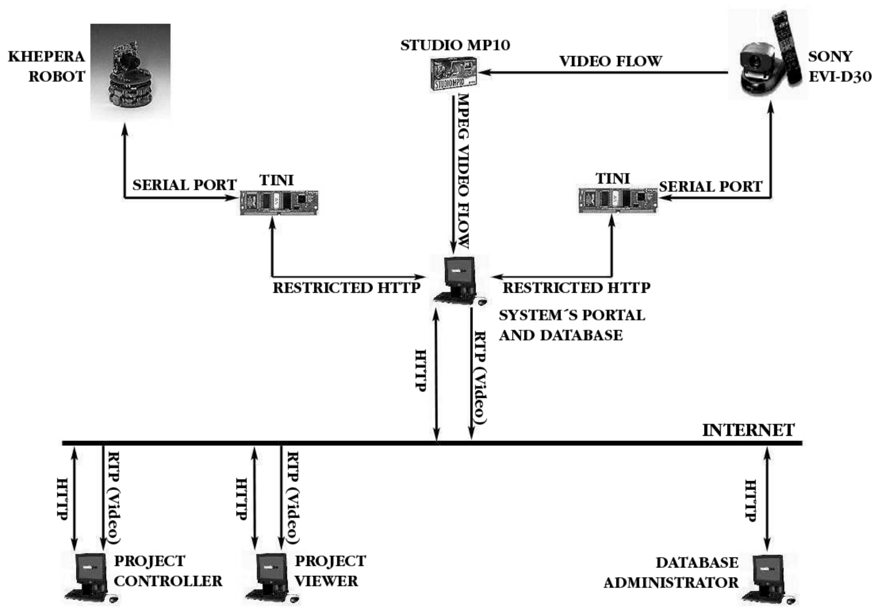

Fig. 3. System's structure

- HypersonicSQL: A Java-Based SQL Database was included for maintenance and to authorize users of the application. This Database is accessed through the portal also.
- Java Communications API (Sun, n.d.*c*): This software is required whenever there is a need to develop an application that includes port's communications.
- Java Media Framework API (Sun, n.d.*b*): This API was used to solve the video transmission issue.

4.3 *Functionality*

The system's functionality can be divided into three interrelated parts:

- *Remote Control*: Providing a remote control access to both the robot and the video camera through WWW to the users is the heart of our system.
- *Video Image*: The transmission and reception of the video image broadcasted by the Sony video camera allows to see what the robot is doing.
- *Database Control*: The database administration is also done through the WWW.

The system provides a basic security policy distinguishing three types of users:

- *Database administrator*: The only one who has access to the *Database Control* part of the system's functionality. It does this by using a Java Applet that communicates with the database through the portal.
- *Project viewer*: It is the user that has the strongest restricted access to the system's functionality. It can only access the *Video Image* part. It does this by either using a Java Applet that communicates with the portal through the RTP protocol or by using one RTP connection provided in the JMF package.
- *Project controller*: This user has access to the heart of the project, that is, *Remote Control* and *Video Image*. It does this by using two Java Applets, one to control the robot and the video camera, and the other to show the live video streaming.

The *remote control* and *database control* part of the system's functionality is conquered by three running Brazil servers, one on each TINI card and the last one in the portal. Table 1 shows the handlers used by the three servers.

The remaining functionality is achieved through a Java Applet that uses the JMF package and the RTP protocol to show the live video stream.

5. CONCLUSIONS AND FURTHER WORK

Throughout this paper a entirely Java-based, fast, compact and cost-effective solution to build Web-based remote control systems has been presented.

It is important to build *micro-servers* that can fit almost anywhere and allow the serviced devices to focus on being effective and have excellent performance without worrying about who is the controlling user, or where this user is physically located.

System's Handlers			
	Handlers	URL	Remarks
TINI1	RestrictClientHandler	/khepera/	Allows only the portal to access this server
	KheperaHandler	/khepera/	Does the real communication with Khepera
	FileHandler	-	Provides certain types of files to remote users
TINI2	RestrictClientHandler	/sony/	Allows only the portal to access this server
	SonyHandler	/sony/	Does the real communication with the video camera
	FileHandler	-	Provides certain types of files to remote users
PORTAL	BasicAuthHandler	/database/	Performs the authentication for the database
	dbcontrolHandler	/database/html/	Sends the database Applet in a HTML document
	AddAuthUserHandler	/database/control /add_user/	Dynamically incorporates new users to the system
	SqlHandler	/database/control/	Controls the Java-based SQL database
	BasicAuthHandler	/project/	Performs the authentication for the project
	controlHandler	/project/html/	Sends the control Applet in a HTML document
	viewerHandler	/project/view/	Sends the view Applet in a HTML document
	historySqlHandler	/project/control/	Dynamically adds the users' commands to the DB
	MultiProxyHandler	/project/control /khepera/	Redirects the users' commands to the Khepera TINI card
	MultiProxyHandler	/project/control /sony/	Redirects the users' commands to the Sony TINI card
	FileHandler	-	Provides certain types of files to remote users
	NotFoundHandler	-	Sends a special Not Found HTML page

Table 1. Handlers implemented in the system's *Brazil* servers

Internet is not the right channel for building real-time systems because of its internal variability in times. Further work could aim to design a real-time based system using another communication channel. In this sense, the new Java real-time specification (Bollella and Gosling, 2000) will help to overcome the characteristic Java limitations.

Future work to adapt the TINI cards to wireless communications (Borkes and Syrisko, 2001) will be an improvement in this type of remote controlled systems, because it allows a new degree of freedom.

6. ACKNOWLEDGEMENTS

The authors would like to thank Denis Bureau (Groupe ESIEE, Paris) and specially Jean-Michel Douin (CNAM, Paris) for the excellent and continuous support throughout the development of the Remote Control System.

7. REFERENCES

Behnke, R. and P.F. Elzer (2001). A user interface for telecontrol of a robot over the internet. *Telematics Applications in automation and robotics* **1**, 547–552.

Bollella, G. and J. Gosling (2000). The real-time specification for java. *IEEE Journal of Robotics and and Automation* **7(4)**, 535–539.

Borkes, J. and H. Syrisko (2001). Trends in wireless. *Telematics Applications in automation and robotics* **1**, 535–540.

DiGiorgio, Rinaldo (n.d.). *Brazil Handlers Web site*. http://www.brazilhandlers.com:9090.

Ietf (n.d.). *RTP Request For Comments*. http://www.ietf.org/rfc/rfc1889.txt.

K-team (n.d.). *Khepera Robot Web site*. http://www.k-team.com/robots/khepera/.

Loomis, D. (2001). *The TINI Specification and Developers Guide*. 1st ed.. Addison-Wesley.

Pinnacle (n.d.). *Studio MP10 documentation*. http://www.dominadm.com/ftp/Manuali/Mp10_eng.pdf.

Rodriguez, F., A. Khamis and M. Salichs (2001). A remote laboratory for teaching mobile robots. *Telematics Applications in automation and robotics* **1**, 307–313.

Roussel, G. and E. Duris (2000). *Java et Internet Concepts et Programmation*. 1st ed.. Vuibert.

Schmid, D., B. Maüle and I. Roth (2001). Performance teletests for industrial robots by the internet. *Telematics Applications in automation and robotics* **1**, 495–499.

Skrtic, S., K. Werthschulte and F. Schneider (2001). Tele-supervision and tele-control of smart homes. *Telematics Applications in automation and robotics* **1**, 547–552.

Sony (n.d.). *EVID30/31 Documentation*. http://rock2000.com/pdf/evid30.pdf and http://www.vision.auc.dk/~tbm/Sony/EVID30.pdf.

Sun (n.d.*a*). *Brazil Web site From Sun Labs*. http://www.sun.com/research/brazil/.

Sun (n.d.*b*). *Java Communications API*. http://www.java.sun.com/products/javacomm/.

Sun (n.d.*c*). *Java Media Framework API*. http://www.java.sun.com/products/java-media/jmf/2.1.1/documentation.html.

Sun (n.d.*d*). *Java Web site - useful for updated documentation*. http://java.sun.com.