USER INTERFACE TECHNOLOGIES FOR HOME APPLIANCES AND NETWORKS

Peter M. Corcoran, Ferenc Papai and Arpad Zoldi
Dept. of Electronic Engineering, University College, Galway

Abstract - **The design and implementation of portable, light-weight user-interfaces to provide access to remote embedded home-systems and home-networks is described. This paper is focussed on emerging standards in the consumer electronics field, including embedded-Java, the handheld device markup language (HDML) and remote frame buffer (RFB) technology as implemented in the public-domain virtual network computer (VNC.) software. A key goal of the present paper is to provide an objective technical assessment of the applicability of each of these technologies to the development of practical consumer applicances and handheld terminals providing enhanced user-access to the home environment.**

1. Introduction

As we approach the 21st century it is clear that a new generation of consumer electronics products is emerging. There new products are increasingly network aware and offer new dimensions in user access. Recent industry initiatives such as the adoption of the *hand-held device markup language* (HDML) by several key industry sectors [1], the announcement of *embedded-Java* [2] and the *remote frame buffer* (RFB) protocol [3] suggest that many new consumer electronics products will support remote access over either a home network, or even over a wide-area-network (WAN) such as the Internet.

As users adopt these new technologies they will come to expect that new appliances may be controlled remotely from multiple access points within the home environment. For example, an end-user might expect to be able to access the home stereo from

(i) a Web-browser on his home-PC, or from

(ii) a Java-phone terminal, or using

(iii) his TV set and its remote control or from

(iv) a personal organizer using a wireless link to the home automation network.

Furthermore, a consumer will expect to achieve this access using an almost identical interface from each of these varied access points. In today's market there are very few suitable interface standards. In consumer markets the closest to a user-interface standard is the InfraRed remote control which has achieved ubiquity in the TV and A/V sectors. At the other extreme of the technology scale the MS-Windows operating systems presents a graphical user-interface standard for home computers. However, between these two extremes of low-tech, low-resource requirements (the IR control) and high-tech, high-resource requirements (the Windows GUI) there is no middle ground.

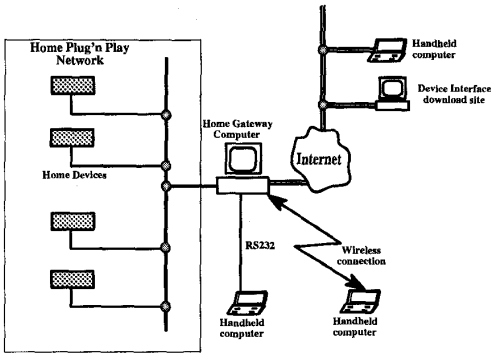

Fig 1: *An overview of the Internet Home: an intelligent gateway brokers between the home network and the Internet; users can access applications running on the gateway using a variety of TCP/IP compatible "application clients".*.

In this paper one of our key goals is to investigate how suitable, lightweight and portable user interfaces can be designed using some of today's key emerging technology standards. The implementation of some example applications is discussed and some of the practical problems involved in handling interfaces across a range of display and access point technologies are illustrated.

2. The Home Network Architecture

In a companion paper [4] we introduce the hypothesis that the Internet-enabled home is likely, in turn, to catalyse the market growth of home networks as manufacturers of domestic appliances and consumer electronic products seek to gain competitive advantage by adding functionality and providing new services via the developing home-Internet infrastructure.

In this emerging market scenario a key issue will be how consumers can access networked home appliances, and equally how these applicances can access services and resources on a TCP/IP wide-area-network (WAN) from a local home network. The former is desirable to improve the accessibility and utility of applicances to end-users in a networked home environment; the latter allows appliances to gain added-value functionality by providing networked based services

Manuscript received June 17, 1998

and by accessing distributed applications software, including systems-level upgrades.

There are two key approaches to these problems:

(i) direct routing of network packets from the home network onto the WAN, and *vice-versa.*

(ii) brokering and management of accesses between the two networks by an *intelligent* gateway.

The gateway architecture described in a companion paper [4] assumes that approach **(ii)** will prevail and descirbes a three-tier software architecture for implementing the home gateway. The thrid software tier is, essentially a user-interface/application layer which supports remote client access to the home gateway over TCP/IP sockets. In this paper we are mainly concerned with how the user-interfaces to such remote clients can be implemented in a platform neutral and portable manner.

2.1 An Overview of the Three Tiers

The physical interface between the gateway and the home network is implemented by a tier-Idriver-agent - *SIOd.* This agent monitors and interprets traffic on the home network. The raw traffic from the home network is then passed on to tier-II which we have implemented as a broker-agent - *CALNetd* - which is responsible for interpreting the home network messages and maintaining a dynamic state-model of the real home network. *CALNetd* is also responsible for interacting with tier-III TCP/IP based client applications and for generating events and placing messages on to the home network via *SIOd.*

In **Fig 2** the relationships between the system software components of the home gateway, the local home network and the wide area network is illustrated. In particular we note that the software components, *SIOd, CALNetd* and any system peer networks or devices (*PNets or PDevs* [4,5]) communicate with one another via TCP/IP network sockets.

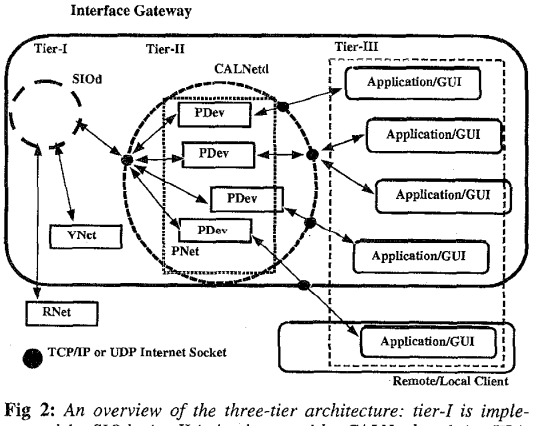

Fig 2: *An overview of the three-tier architecture: tier-I is implemented by SIOd; tier-II is implemented by CALNetd and tier-III is the TCP/IP client application layer.*

In this configuration, which is employed in our prototype gateway, all of the main components run on the same *interface gateway* - in our case a standard PC. Applications may run on other *application clients* other networked PCs, but in this implementation of the gateway architecture the key system software components run, as independent *daemon* processes on a single PC. This is illustrated in **Fig 3** which shows that, although client applications may have their user-interface accessed directly from the gateway, it is more likely that the user-interfaces will run on separate, *applications client* hardware connected to the TCP/IP network infrastructure.

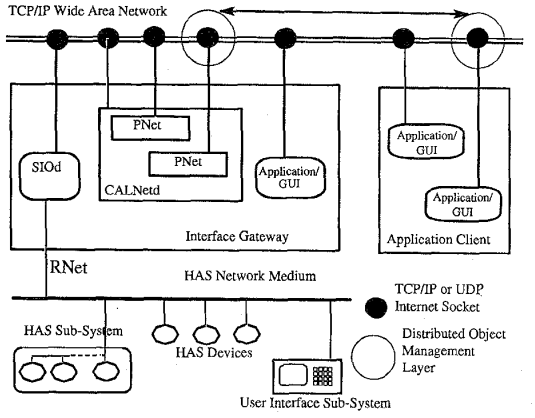

Fig 3: *Diagramatic representation of the relationships between system software components, and the physical components of the Wide Area Network and the Home Network.*

3. User-Interface Technologies

In creating the user interfaces to access a Home Network two main requirements are defined. First, the portability of the user interfaces, allowing access to the home network from a wide range of hardware platforms and operating systems. The other principle requirement for the interfaces is that they should be as thin as possible to facilitate dynamic uploading and operation on hand-held computers and resource constrained devices. We next present a brief summary description of three key approaches to the problems outlined above: Personal/Embedded Java, HDML and remote access to user interfaces using the RFB protocol.

Personal/Embedded Java is an application development enironment for creating software for embedded devices and resource constrained environments. Similar to Java, embedded Java consists of core and standard extension APIs, more precisely Embedded Java includes a feature level subset of Java, therefore Embedded Java applications are upward compatible with Java. If Embedded Java is used, the interface is uploaded to the remote access client as executable Java byte-code and executed in a *Java virtual machine* (JVM) on the client. However this can be resource-

heavy, particularly at the client end as a full implementation of the JVM is required and this, typically, requires 8-16M of memory.

The Hand-Held Device Markup Language (HDML) provides an efficient markup language for wireless and other hand-held and resource-constrained devices. Its focus goes beyond presentation and layout, providing and explicit navigation model which does not rely upon the visual context required of HTML. HDML is an efficient mean to provide content via the WWW infrastructure to hand-held devices such as cellular phones, pagers and wireless PDA's. In this second approach a HDML interface is loaded by a HDML interpreter - somewhat analogous to a Web browser - and events are generated in the remote server application using the well-known common gateway interface (CGI).

The Remote Frame Buffer (RFB) protocol is a simple protocol for remote access to graphical user interfaces. This protocol is truly a "thin-client" protocol: it has been designed to make very few requirements of the client. In this way, clients can run on the widest range of hardware, and the implementation of a client is kept as simple as possible. This protocol also makes the client stateless, and because of this stateless nature of the client the connection can be closed at any time by either of the sides without any system-level consequences. Because of the design of the RBF protocol writing a viewer for a client is a very simple task, and RFB clients may function optimally on small hand-held appliances.

4. Java and Related Technologies

We have previous experience in implementing home-networking solutions in Java [5, 6, 7] so it was natural to begin by investigating the potential of a pure Java solution to provide an answer to our user-interface requirements. Indeed it is possible to implement quite sophisticated user-interfaces using Java, while maintaining the core application on an independent server using servlet technology.

Unforturnately to run a Java virtual machine (JVM) at the client-end will still require 4-8M of memory. During the course of our work a low-resource implementation of *Personal Java* was announced which was built on the *Inferno* operating-system, however this had not become available in time to allow us to evaluate this form of Java solution to our UI requirements.

We conclude that, at present, Java does not offer a very useful solution for the lightweight user-interface requirements of the consumer electronics sector, although it does have strong merits in middle to high-end Internet Appliance solutions. We predict that Java will become much more applicable to consumer electronic applications once low-cost specialized Java CPU units become available for embedded applicances.

In **Fig 4** we show how the Java-based user interface for a digital photography application can be accessed via a conventional web browser. This provides a high level of portability for the user interface, but at the expense of requiring adequate memory resources on the host hardware to run a modern web browser.

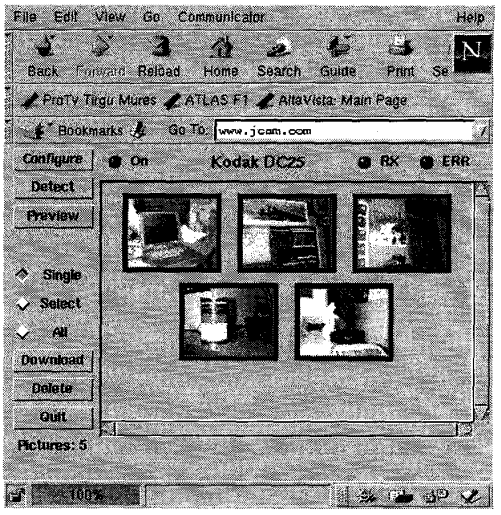

Fig 4: *A demonstration of the user-interface to a generic digital photography application; this Java-based UI is running in the JVM of the Netscape web browser.*

5. HandHeld Device Markup Language

The Hand-held Device Markup Language (HDML) provides an efficient markup language used to create hypertext-like content for wireless and other hand-held and resource-constrained devices. This characteristic is a very important one, as the target of our user interfaces usually are resource constraint devices. These resource constraints could be:

- small display area (size)
- limited input capabilities
- limited network bandwidth
- limited hardware resources

The World Wide Web provides a robust, flexible and ubiquitous model for accessing information. The very wide acceptance and adoption of the WWW as the preferred means of disseminating and accessing information from desktop PCs and workstations has created a demand for access to the same information for other devices. These devices or "alternate platforms" range from voice and fax-based user agents to low-cost Network Computers and to handheld devices such as mobile phones to PDAs.

While the web infrastructure and the afferent protocols fully support these alternate platforms, HTML itself does not. In particular the navigation and display models inherent to HTML collapse when applied to a

typical handheld device. However, combining the use of standard web protocols and infrastructure (URLs, HTTP, SSL plus CGI, Perl, and commercial web servers) with an alternate but complementary markup language, allows handheld devices to function as fully fledged web-clients. For our projects, this means to combine existing and functional web-technologies for interacting with users with a small and less resource-heavy interface on the client side.

The focus of HDML goes beyond presentation and layout, providing an explicit navigation model which does not rely upon the visual context required of HTML. HDML is an efficient mean to provide content via the WWW infrastructure to hand-held devices such as cellular phones, pagers and wireless PDA's.

In the context of our home gateway a HDML interface is generated at the gateway/server-end and is loaded by a HDML interpreter on the client. The HDML interpreted is somewhat analogous to, but more light-weight than, a conventional Web browser. Events are generated in the remote server application using the well-known common gateway interface (CGI).

Like many other langauges, HDML requires a run-time environment to be useful. To make HTML to be functional we need in this run-time environment an interpreter and a viewer, analogous to a web-browser. A difficulty here is that reference implementations of HDML browser/interpreters do not exist for very many platforms. It would appear, at this time, that HDML is likely to remain a niche standard focussed primarily on mobile telephone markets.

6. Remote Frame Buffer Technology

In this section we describe the *remote frame buffer* (RFB) technology which was recently released by Oracle research laboratories under the GNU Public Licence. This RFB technology forms the core of their *virtual network computer* (VNC) application.

Our interest in RFB is in its potential application as a user-interface delivery mechanism in the context of the home-gateway architecture described in section 2. In this scenario the system software components run mainly on the home-gateway computer, except the client interface which is uploaded to an application client.

The most important component of the system is the main server module. This module must be able to communicate with both a home network and an application client using TCP/IP sockets. The beauty of RFB is that, like our gateway architecture, it is very "socket-oriented" and compliments the *SIOd* and *CAL-Netd* implementations described earlier.

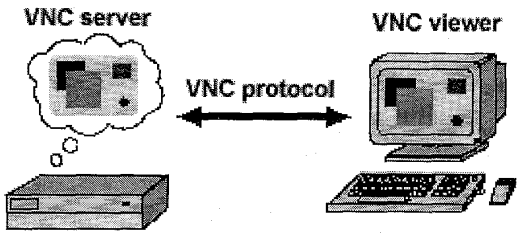

Fig 5: *An overview of the virtual network computers; the RFB protocol is used to export the entire desktop from the server; at the client-end the desktop state is cached in a frame-buffer and redrawn as required; events are relayed back to the server from the client and allow users to interact directly with applications on the server..*

6.1 The RFB Protocol

The RFB protocol is a simple protocol for remote access to graphical user interfaces. It is based on the concept of a remote framebuffer (RFB). The protocol simply allows a server to update the framebuffer displayed on a viewer. Because it works at the framebuffer level it is potentially applicable to all operating systems, windowing systems and applications. This includes X/Unix, Windows and Macintosh, but can also include PDAs. The protocol will operate over any reliable transport such as TCP/IP.

This is truly a "thin-client" protocol: it has been designed to make very few requirements of the viewer. In this way, clients can run on the widest range of hardware, and the task of implementing a client is made as simple as possible.

6.2 Rectangular updates

The display side of the protocol is based around a single graphics primitive: "put a rectangle of pixel data at a given x, y position". This might seem an inefficient way of drawing arbitrary user interface components. However, as there are a variety of different encoding schemes for the pixel data, an appropriate scheme can be selected for each rectangle that is sent, in order to make the most of network bandwidth, client drawing speed and server processing speed.

The lowest common denominator is the so-called raw encoding, where the rectangle is simply pixel data sent in left-to-right scan-line order. All clients and servers must support this encoding. However, the encoding actually used on any given VNC connection can be negotiated according to the abilities of the server, the client, and the connection between the two.

The copy rectangle encoding, for example, is very simple and efficient and can be used when the client already has the same pixel data elsewhere in its framebuffer. The server simply sends an x, y coordinate giving the position from which the client can copy the rectangle of pixel data. This means that operations such as dragging or scrolling a window, which involve substantial changes to the screen, may only require a few bytes. Most clients will support this encoding,

since it is generally simple to implement and saves bandwidth.

To understand better the reasoning behind the RFB protocol consider a PC desktop which typically has large areas of solid color and of text. Some of the most effective encoding takes advantage of this by efficiently describing rectangles consisting of a majority (background) color and 'sub-rectangles' of different colors. There are numerous other possible schemes. We might use a JPEG encoding for still images or MPEG for efficient transmission of moving images.

6.3 Adaptive update protocol

A sequence of such rectangles makes a framebuffer update (or simply update). An update represents a change from one valid framebuffer state to another, so in some ways is similar to a frame of video, but it is usually only a small area of the framebuffer that will be affected by a given update. Each rectangle may be encoded using a different scheme. The server can therefore choose the best encoding for the particular screen content being transmitted and the network bandwidth available.

The update protocol is demand-driven by the client. Thus, the server only sends an update in response to an explicit request from the client. This gives the protocol an adaptive quality. The slower the client and the network are, the lower the rate of updates becomes. Each update incorporates all the changes to the 'screen' since the last client request. With a slow client and/or network, transient states of the framebuffer are ignored, resulting in reduced network traffic and less drawing for the client. This also improves the apparent response speed.

6.4 Input protocol

The input side of the protocol is based on a standard workstation model of a keyboard and multi-button pointing device. The client sends input events to the server whenever the user presses a key or pointer button, or whenever the pointing device is moved. These input events can also be synthesized from other non-standard I/O devices.

6.5 Connection Setup and Shutdown

When the connection between a client and a server is first established, the server begins by requesting authentication from the client using a challenge-response scheme. The server and client then exchange messages to negotiate desktop size, pixel format, and the encoding schemes to be used. The client then requests an update for the entire screen, and the session begins. Because of the stateless nature of the client, either side can close the connection at any time without adverse consequences.

6.6 Practical Demonstration of RFB

Some active demonstration of the RFB protocol in action are shown in **Figs 6 & 7** below. Basically we modified a typical home-network application - in this case a picture downloader for digital still cameras - to run with a more compact user-interface than it would normally have on a desktop PC - see **Fig 4** for details.

In **Fig 6** we show the application after thumbnails from several pictures have been downloaded from a camera. We can see that the user interface for the application is "echoed" on the screen of a Palm Pilot PDA which is running an RFB client program. The actual application is running on the PC, but can be easily controlled from the PDA.

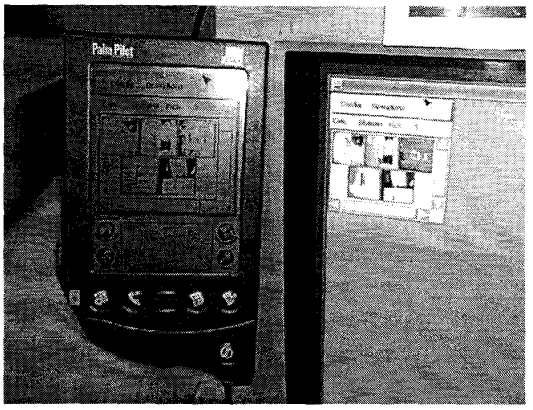

Fig 6: *A demonstration of the RFB protocol running on a Palm Pilot PDA. The original application, can be seen ion the screen of a desktop PC which is used to emulate a home-gateway server.*

In **Fig 7** the same modified application is shown, but note in this case that the menu is active. This event can be initiated from either the remote client, or from the server PC itself. Also note that the PDA has been removed from its communications cradle in **Fig 7** - the RFB protocol is stateless and when the PDA is returned to its cradle if will resume communications with the RFB server as if the communictions link had not been broken.

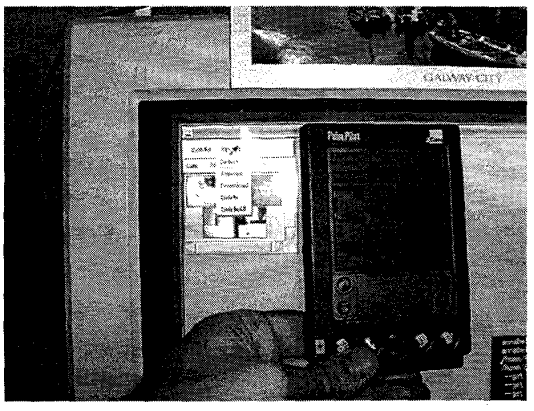

Fig 7: *Another demonstration of an RFB client running on a Palm Pilot; note that the menu of the application is now active.*

7. Experimental Network Infrastructure

In order to evaluate and implement various prototypical user interface solutions we needed an experimental network infrastructure which mirrors the infrastructure in which a home gateway will operate. To this end we have a small CEBus powerline network available in our laboratory and several other networking technologies were available to emulate an external wide-area network. These included ATM network-interface hardware and an ATM switch [8]. A local ethernet network segnment was also available.

To emulate the home-gateway we use standard x86 PC running a Unix clone OS. This allows us to run the *SIOd* and *CALNetd daemon* programs which we have described in earlier work [6, 7, 8]. In a practical application we expect that a set-top box or embedded gateway would replace this experimental gateway solution. The PC is connected to a typical power-line home network and in this way can provide access to networked consumer appliances. Connections to other home networks, including USB and IEEE 1394 devices may be readily achieved by adding adapter cards to the standard PC. This Home Gateway can also communicate with hand-held computers and PDAs using serial-line PPP connections.

8. Results & Conclusions

We have tested various combinations of each of the technologies described in this paper in conjunction with our experimental gateway architecture, described in detail in a companion papers [4]. Some examples of what we consider to be typical home applications have been described above. In this section we wish to focus on the practical experiences and to outline the applicability of each of the technology standards described in this paper to the design of practical consumer electronic applications and embedded appliances.

We have previous experience in implementing home-networking solutions in Java [5, 6, 7] so it was natural to begin by investigating the potential of a pure Java solution to provide an answer to our user-interface requirements. Indeed it is possible to implement quite sophisticated user-interfaces using Java, while maintaining the core application on an independent server using servlet technology. Unfortunately to run a Java virtual machine (JVM) at the client-end will still require a minimum of 4-8M of memory.

During the course of our work a low-resource implementation of *Personal Java* was announced which was built on the *Inferno* operating-system, however this had not become available in time to allow us to evaluate this form of Java solution to our UI requirements. Thus our conclusion is that, at present, Java does not offer a very useful solution for the lightweight user-interface requirements of consumer electronics appliances, although it does have strong merits in middle to high-end Internet-Appliance (IA) solutions. We predict that Java will become much more applicable to consumer electronic applications once low-cost specialized Java CPU units become available for embedded applicances.

HDML is an efficient mean to provide content via the WWW infrastructure to hand-held devices such as cellular phones, pagers and wireless PDA's. In the context of our home gateway a HDML interface can be generated at the gateway/server-end and loaded by a HDML interpreter on the client. The HDML interpreted is somewhat analogous to, but more lightweight than, a conventional Web browser. Events are generated in the remote server application using the well-known common gateway interface (CGI).

Like many other langauges, HDML requires a run-time environment to be useful. To make HTML to be functional we need in this run-time environment an interpreter and a viewer, analogous to a web-browser. A difficulty here is that reference implementations of HDML browser/interpreters do not exist for very many platforms. Thus we have worked mainly with HDML browsers/interpreters on conventional desktop PC systems. Although quite lightweight browser implementations can be realized in this way, it is not clear if there will be enough broad support for HDML as a independent standard. If this does not happen then it is unlikely that it will be supported across a broad range of *application client* platforms. Thus, we conclude that at present HDML is most likely to remain a niche standard focussed primarily on mobile telephone markets.

The *remote frame buffer* (RFB) technology was recently released by Oracle Research Laboratories (UK) under the GNU Public Licence. This is an interesting event, following as it does on the release of source code by Netscape [9], and it guarantees that RFB will be adopted widely in many applications and is also likely to enter into a rapid development cycle [10].

Our interest in RFB is in its potential application as a user-interface delivery mechanism in the context of our home-gateway architecture [4]. In this scenario the system software components run mainly on the home-gateway computer, except the client interface which is uploaded to an application client. The beauty of RFB is that, like our gateway architecture, it is very "socket-oriented" and compliments the *SIOd* and *CAL-Netd* implementations described earlier. This, in turn, provides a *de-facto* compliance with existing TCP/IP networks and Internet standards. For these reasons we feel that RFB offers an important and very significant means to implement portable user interfaces for consumer electronic applications.

Ultimately, the application of Java technology will have the greatest and most long-term impact on the

consumer electronic markets, but in its present state of maturity the technology is too costly and too resource intensive to find applications outside a limited niche market. However, as low-cost hardware solutions implementing the JVM become available we should see a broader application of Java as a key user-interface technology.

References

[1] http://www.uplanet.com/pub/

[2] http://www.javasoft.com/

[3] http://www.orl.co.uk/vnc/

[4] Corcoran P. M., "Mapping Home Appliances to TCP/IP Sockets using a Three-Tiered Home Gateway Architecture.", *IEEE Internetional Conference on Consumer Electronics,* Los Angeles, June. 1998.

[5] Corcoran, P.M., Desbonnet, J. and Lusted, K. "CE-Bus Network Access via the World Wide Web", *IEEE Trans. Consumer Electronics*, Aug. 1996.

[6] Desbonnet, J and Corcoran P. M.., "System Architecture and Implementation of an Internet/CEBus Gateway", *IEEE Transations on Consumer Electronics*, p1057-1062, Nov. 1997.

[7] Corcoran P. M. and Desbonnet, "Browser Style Interfaces to a Home Automation Network", *IEEE Transactions on Consumer Electronics,* p1063-1069, Nov. 1997.

[8] Cucos A. and Corcoran P. M., "Real-Time ATM for Remote Access to Home Automation and Digital Home A/V Networks", *IEEE Internetional Conference on Consumer Electronics,* Los Angeles, June. 1998

[2] http://www.mozilla.org

[3] http://www.ssc.com/linux/Eric/cathedral.html

Biography

Peter Corcoran received the BAI (Electronic Engineering) and BA (Maths) degrees from Trinity College Dublin in 1984. He continued his studies at TCD and was awarded a Ph.D. in Electronic Engineering in '87 for research work in the field of Dielectric Liquids. In '86 he was appointed to a lecturship in University College Galway. From '94 to '96 he worked as general manager in the electronics division of the first Sino-Irish Joint Venture, based in Beijing, PRC. During '96 he was also a visiting Professor in Telecommunications at the Tech. Univ. of Cluj-Napoca, Rommania. His research interests include consumer electronics, embedded computing, networking, instrumentation and telecommunications technologies. He is a member of the IEEE.

Ferenc Papai received his B.S. degree in Mathematics and Informatics from "Petru Maior" University Tirgu-Mures, Romania, in 1997. Currently he is completing the M.S. degree in Applied Science at University College Galway, Ireland.

Arpad Zoldi received his B.S. degree in Mathematics and Informatics from "Petru Maior" University Tirgu-Mures, Romania, in 1997. Currently he is completing the M.S. degree in Applied Science at University College Galway, Ireland. His major interests include OOP and Java programming, Internet technologies, networking, operating systems, distributed computing.