Using Mobile Code Interfaces to Control Ubiquitous Embedded Systems

Kari Kangas and Juha Röning
University of Oulu

Using Mobile Code Interfaces to Control Ubiquitous Embedded Systems

Kari Kangas and Juha Röning

University of Oulu, Department of Electrical Engineering
Computer Engineering Laboratory
P.O. BOX 4500, FIN-90401 Oulu, Finland
{macconen, jjr}@ee.oulu.fi

Abstract

Devices controlled by embedded computers are becoming an integral part of our everyday life, as processor and memory capacities continue to increase while their cost decreases. In some embedded systems, however, the limited input and output capacities are beginning to restrict the design of complex functionality. Furthermore, as wireless communication devices are becoming commonplace even in embedded systems, the communication and interoperation between different systems will be increasingly important in the future. This paper describes a flexible, yet powerful concept that explains how a mobile code can be conveniently utilized by mobile users to control ubiquitous and diverse embedded systems in different environments. Apart from providing flexibility, the concept also aims to keep the embedded systems as simple as possible. We will illustrate the concept by presenting as an example a virtual user interface for a videocassette recorder. We will also discuss the possible benefits and drawbacks of the system. The concept described here can be extended to allow the mobile code to be used as interconnecting "glue" in diverse embedded systems. This glue could connect systems from several manufacturers to create smart environments that can be controlled by a single simple device.

1. Introduction

This paper describes the use of a mobile code to create Virtual User Interfaces (VUIs) that can be used to conveniently control ubiquitous embedded systems. Diverse embedded systems are becoming an integral part of our everyday life. We already encounter a large number of embedded systems not only at home but also elsewhere in our daily life. In the near future, embedded systems will be used to make up smart environments; i.e. environments that contain a very large number of objects controlled by a hidden, or embedded, computer [1]. In such environments, it is desirable that the same underlying technology and a single simple device can be used to control several embedded systems. We do not want every system to require a separate remote control unit or to include a complete user interface, as is usually the case at the present. Furthermore, the same technology should also be available to a mobile user. The mobile user should be able to control the unknown embedded systems that he or she may encounter in various environments. In addition, the communication between the embedded systems and the controlling device should be as local as possible, so that low-power and high-bandwidth communication devices, such as Bluetooth [2], could be used. Internet connection should only be required in exceptional situations. Local communication is closely related to the principle of self-containment. The embedded system in itself should contain at least the most important components required for controlling it. In this paper, we describe a concept that is our first step towards a system that can be used to control truly ubiquitous embedded systems in future mobile computing.

The functionality of embedded systems has traditionally been restricted by the high cost of processor power and memory capacity [3]. Low cost is especially important in consumer electronics. As the cost of powerful processors, memory, and other microelectronics has dropped dramatically, a large variety of extended functionality is being designed for embedded systems [4]. However, input and output capabilities have not improved at the same pace as the other capacities and some cases where complex functionality is implemented even require design compromises.

Embedded systems, such as Videocassette Recorders (VCRs) and microwave ovens, usually have displays that provide only very limited functionality for displaying text and high-resolution graphics. These displays are usually constructed by using Light Emitting Diodes (LEDs) or Liquid Crystal Displays (LCDs). Furthermore, these systems usually contain limited input devices, such as miniature keyboards, that may be inconvenient to use. Cost is the main reason why high-quality interfaces cannot be included in all embedded systems. However, this is not necessarily the only reason. Display and keyboard size is strictly limited in devices such as cellular phones or personal heart rate monitors.

Thus, the input and output capabilities are becoming the limiting factor in embedded systems in the future. This is especially problematic in the field of consumer electronics where most devices now fulfil their basic functionality requirements, and extended features are included to attract consumers. Setting up a VCR to record a TV show at a desired time without first consulting the manuals is already too difficult for some users. As embedded devices are becoming increasingly ubiquitous in the future, there will be an even more diverse user population while at the same time the necessity of being able to control various embedded systems will increase. To make the new users comfortable with new systems, we will need descriptive error messages in plain text, context-sensitive online help facilities, and automated configurations instead of cryptic error codes and signals.

Creating a virtual user interface by using the capabilities of an external control device (also referred as a client), such as a Personal Digital Assistant (PDA), can provide abundant input and output capabilities for an embedded system. In addition, the same device can be used to create virtual user interfaces for various systems, and it therefore also provides a notably cost-efficient solution compared to the alternative of including these input and output capabilities in each individual embedded system.

This paper describes a technique for creating virtual user interfaces by using a mobile code. For the purpose of this paper, mobile code is a software component that comes from one computer system and is executed by another. Java applet is an example of such a mobile code component. For the mobile code to be utilized, the embedded system must contain the code that is requested and used by an external control device to create a virtual user interface. This virtual user interface is then used to control the embedded system. The virtual user interface and the embedded system communicate by using an internal protocol that is completely invisible to the outside system. Below, a PDA will be used as an example of a control device.

The mobile code technique has three main goals. The first and most important goal is to minimize the processor and memory requirements of an embedded system while still providing enough flexibility and scalability to allow the technique to be used in diverse applications. This is achieved as follows. The software in the embedded system can be constructed by using traditional programming conventions and languages. The embedded system is not required to contain a virtual machine or to use software written in any specific programming language. In addition, different system manufacturers only need to agree on a very simple mobile code protocol. The only purpose of this protocol is to help the PDA user to identify different systems and to transfer the mobile code. Furthermore, the technique presented in this paper does not place any restrictions on the actual mobile code that is used to create a virtual user interface, nor does it restrict the appearance of the virtual user interfaces.

The second goal is to make it possible to construct self-contained embedded systems that can be controlled by using only local communication between the embedded system and the PDA. This would make it possible to use only low-power and high-bandwidth devices for communication. Local communication is achieved by storing the mobile code in each individual embedded system. It should be noted that the flexibility mentioned above is not sacrificed, as some parts of the mobile code may come, for example, from a server in the Internet. Storing parts of the mobile code in an external server violates the self-containment principle, but may be beneficial in some situations.

The third goal is to enable mobile computing. In mobile computing, the user roams in different environments and may encounter embedded systems previously unknown to him or her. It is therefore important that the PDA can adapt to new environments and new systems. In addition, the communication between the embedded systems and the PDA may be sporadic and the connections highly volatile. Adaptation to new systems is achieved by downloading the mobile code from each embedded system. The PDA is not required to identify the embedded system, but only to be able to execute the mobile code. A connectionless message based communication protocol is used to match the volatile communication channels.

This paper has been organized as follows. Chapter 2 will present related work. Chapter 3 will describe in detail the concept of utilizing a mobile code to create virtual user interfaces. Chapter 4 will describe an example system constructed to demonstrate how the mobile code concept can be implemented. Chapter 5 will discuss the benefits and drawbacks related to the mobile code technique. Chapter 6 will draw a conclusion and illustrate future work.

2. Related Work

Creating remote displays by using the X Window protocol in Unix or in systems utilizing teleporting [5] can be considered similar to creating virtual user interfaces. To make a clear distinction between these systems and the virtual user interface concept presented in this paper, such systems are said to utilize a display protocol. The systems utilizing a display protocol are constructed so that an embedded system executes a program and a PDA displays a user interface of that program by using its display capabilities. These two systems share a common display protocol that is used to exchange display information. The PDA uses the information sent by the embedded system to construct and modify the user interface. When the user performs an action using the remote user interface, the PDA sends data containing the user commands to the embedded system for processing.

As obvious, designing a general-purpose display protocol usable in every possible application is by no means a trivial task. For example, the protocol must be flexible enough to meet the requirements of the different applications. Flexibility may, in some situations, mean that the protocol will be very complex and difficult to implement and use. Flexibility is usually achieved by including a redundant code to be utilized in future applications, and this redundant code may require too much storage to be usable in some embedded systems. Furthermore, the protocol must only require a small amount of communication bandwidth to be usable in systems with very limited communication resources. Even if such a general-purpose protocol can be designed, it must be widely accepted by the competing industry and be standardized. This can be very difficult, especially in the highly dynamic field of consumer electronics.

One way to reduce the required communication bandwidth is to use a lightweight version of the display protocol to create a remote user interface. An example of such a protocol is the Virtual Network Computing (VNC) protocol [6]. These protocols usually differ from normal display protocols in that the level of message abstraction is raised in order to reduce the amount of data transmitted between the embedded system and the PDA. In other words, whereas normal display protocols usually describe the remote user interface in great detail, lightweight protocols only describe the overall structure and rely on the more complex display services provided by the remote computer. This naturally reduces the need for communication but also reduces flexibility.

Another drawback in designing a standard communication protocol is that the embedded system generates the protocol messages that are used by the PDA to display the user interface. This means that the command messages arriving from the PDA as a result of user interaction must be translated into corresponding user interface messages. These messages are then sent back to the PDA. Assuming that the embedded system is relatively simple, the resource requirements for keeping track of the remote user interface status and generating correct messages may increase to undesirable extend.

One way to reduce the bandwidth requirement of remote user interfaces is not to use a display protocol at all, but to use some other relatively lightweight and possibly ubiquitous and well-known communication protocol instead. One possibility is to use the Hypertext Transfer Protocol (HTTP), usually combined with Common Gateway Interface (CGI) programs [7, 8]. The PDA requests Hypertext Markup Language (HTML) pages from the embedded system and displays them using a web browser. CGI programs can be used to construct these pages dynamically, so that they reflect the current status of the system. In addition, CGI programs are used to receive the commands issued by the web browser user. The benefit in this approach is that ubiquitous web browsers can be used to control the embedded system. Furthermore, the utilization of HTML pages and CGI programs is a simple and well-known technique. The drawbacks include the requirements of the embedded system to execute a TCP/IP stack and at least a minimal HTTP server. In addition to this, HTML pages have several limitations when used to create remote user interfaces. First of all, HTML pages cannot display data with temporal characteristics by default i.e. an HTML page can only be used to display a static snapshot of the system status. One way to display the dynamics is to use the HTTP push technique, but it may, in some circumstances, take up too much communication bandwidth. As an example, consider a situation where a new HTML page is pushed to the PDA every second, or every tenth of a second.

Another and more flexible way to create remote user interfaces is to use the HTTP and CGI programs in combination with Java applets, as outlined in [9]. The PDA uses the HTTP to request a Java applet. When the Java applet is run, it communicates with the embedded system using an application-specific protocol. The use of HTTP and Java applets naturally requires the embedded system to contain a communication component in addition to an HTTP server. The communication component communicates with the Java applet using an application-specific protocol. HTTP can also be used for communication between the Java applet and the embedded system, using CGI programs as described earlier. In essence, the principle in the HTTP and Java applet combination is the same as in the mobile code approach presented in this paper: to use a commonly agreed protocol to obtain a mobile code from the embedded system and then to use this code to create a remote user interface. An application-specific protocol is then used for subsequent communication between the remote user interface and the embedded system. The main problem with HTTP is that it is usually used in combination with the TCP/IP protocol. Most of the embedded systems in smart environments can be expected to be relatively simple; thus the TCP/IP protocol may be too heavyweight for such systems if they do not otherwise require an Internet connection.

The commercial systems Inferno [10] and Jini [11] also offer a solution for controlling networked embedded systems, and the control aspects of Jini are very similar to the HTML and Java applet approach described above. However, both Inferno and Jini rely heavily on running a virtual machine in the embedded system and use a restricted set of programming languages. Furthermore, as they have both been designed by default to operate by using an Internet connection and the TCP/IP protocol, they would require slight customization to be suitable for wireless mobile computing. The reader should note that both Inferno and Jini provide a rich set of features other than control interfaces, which means that placing a virtual machine in an embedded system may be suitable in some applications. However, we feel that the manufacturer of the embedded system should not be forced to use a specific virtual machine or programming language.

Systems utilizing a mobile code for controlling embedded systems have also been described in [12] and [13]. Both of these systems rely on mobile code servers and the Internet for obtaining the user interface code. We feel that the possibility to construct self-contained embedded systems is so important that it justifies the hardware overhead in our approach compared to these systems.

3. Virtual User Interface as a Concept

A system utilizing mobile code virtual user interfaces is presented schematically in Figure 1. The main requirement for the complete system is that the embedded system and the client (e.g. a PDA) are connected via a communication channel of arbitrary form. This channel can be implemented by using an Infrared (IR) or wireless radio link or a cable. The channel is used for communication between the embedded system and the client. The communication includes the creation of a virtual user interface and the subsequent communication between the virtual user interface code and the embedded system. The client requests the mobile code from the embedded system by using a mobile code protocol. Both the embedded system and the client agree on the format of this protocol. The mobile code protocol provides only very limited services. These services can be used to identify the basic parameters of the selected system and to transfer the mobile code. Messages transferred through the communication channel that do not match the mobile code protocol can be considered to follow an internal communication protocol. This internal protocol is used between the virtual user interface and the control module in the embedded system, and it can be relayed directly through the mobile code protocol.

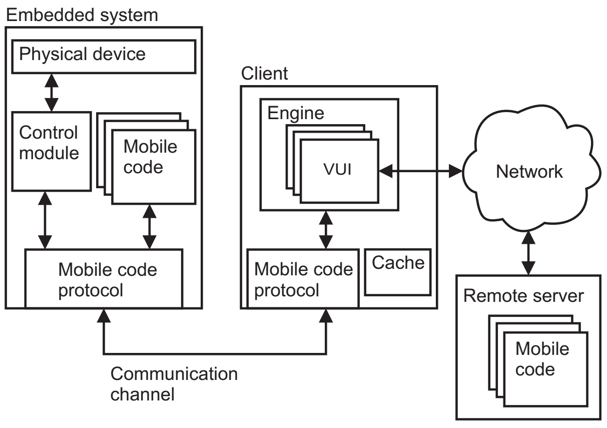

Figure 1. An outline of a system utilizing mobile code to create virtual user interfaces.

In addition to the component providing the mobile code protocol, the embedded system consists of a mobile code and a control module. The mobile code is stored on a non-volatile storage medium, such as an Electrically Erasable Programmable Read Only Memory (EEPROM) or Flash ROM, and sent to the client when required. The client then executes the code to create a virtual user interface. The embedded system may contain different types of mobile code to be sent to different types of clients. The amount of mobile code that can be stored in any particular embedded system depends on how much non-volatile storage can be afforded or feasibly included.

The control module acts as a communication relay between the virtual user interface in the client and the actual physical embedded system. The control module typically receives command messages, such as user requests to start VCR playback, originating from the virtual user interface. It transforms these commands into internal signals and sends them to the physical device. The control module also receives status signals from the physical device. The status signals reflect the effects of the commands issued and other external events, such as error conditions. The control module transforms these signals into status messages and sends them to the virtual user interface in the client. These command and status messages form the private communication protocol between the control module and the virtual user interface. The protocol can be chosen to match the exact requirements for each particular application. In other words, this internal communication protocol is likely to vary from one application to another. However, the system designer could also use a standardized communication protocol as an internal protocol; provided a suitable protocol is available.

The client consists of a component providing the mobile code protocol and a mobile code execution engine. The engine executes the mobile code and provides an execution environment for it. The engine provides an abstraction layer of the actual client by providing basic services for the mobile code. As an example, such services may include a communication channel that can be used to communicate with the embedded system where the mobile code originated. Different engines may offer different services and provide varying levels of abstraction. As an example, even the client's operating system can operate as an execution engine for the mobile code. This requires that the mobile code is a complete executable binary image for that particular operating system.

It is also worth noting that the embedded system can contain only parts of the mobile code required to create a virtual user interface. In other words, some parts of the mobile code can be obtained from other code sources than the embedded system. Remote mobile code servers can operate as such code sources. Whether the mobile code is obtained fully from the embedded system or partly from the remote code server can be invisible to the client. This can be achieved by designing the mobile code in the embedded system to operate as a bootstrap code. This bootstrap code downloads the rest of the mobile code from the remote code server. This technique is suitable in situations where the total size of the mobile code is very large, or some parts of the mobile code are rarely required in everyday use. In an extreme situation, the embedded system may only contain a few lines of bootstrap code that is used by the mobile code engine to download the actual code from the remote server.

In order to minimize the need to constantly transfer the mobile code from the embedded system to the client for execution, mobile code caching can be utilized. For this purpose, the mobile code protocol may include a service that can be used to query the time when (if ever) the mobile code stored in the embedded system was last changed.

The technique described earlier in this Chapter can be considered to follow both the code-on-demand (COD) and the remote evaluation (REV) paradigms [14]. Both the embedded system and the client can initialize the creation of a virtual user interface. When the client initiates VUI creation, as is usually the case, the COD paradigm is followed. When the embedded system initiates VUI creation, the REV paradigm is followed. An example of VUI creation that is initialized by the embedded system is a situation where the embedded system uses a virtual user interface to notify the client of an exceptional situation. Such a situation could be the outcome of a scenario where the embedded system has detected a minor hardware failure during a self-check and wants to notify this to the first client that establishes a communication channel with it. To illustrate this with a concrete example, a smoke detector could notify a maintenance person walking by of a battery low condition.

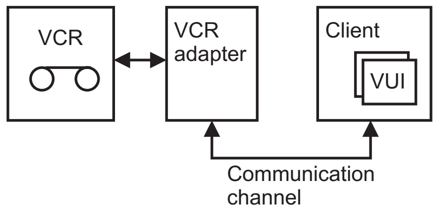

Figure 2. An example system providing a virtual user interface for a VCR device.

4. Implementation of a Virtual User Interface System

This Chapter explains how the concepts described in the previous Chapter can be utilized in an actual system. The example system provides a virtual user interface for a simple VCR device. This system was constructed to gain deeper understanding of the virtual user interface and the mobile code as a problem domain. The reader should note that the example system utilizes only a small fraction of the possibilities implicit in the mobile code approach. Furthermore, we agree that the VCR is not a good example in a sense that most of the current VCRs use also a TV screen to output status information.

All the software components in the system were constructed using the Java programming language. The main reason for this was the desire to be able to run the example system in a variety of hardware platforms. In a real system, the program run in the embedded system would probably be constructed by using a low level programming language, such as C or Assembler. The example system is presented schematically in Figure 2.

The system logically consists of four components: an actual VCR device, a VCR adapter, a communication channel, and a client system. It was difficult to find a VCR that could provide suitable feedback for the VCR adapter, mainly because the current VCRs usually provide only visual status signals. We can easily issue commands to the VCR via an infrared transmitter, but there is the risk that the VCR and the adapter enter an inconsistent state when, for example, the tape reaches the end. This was the reason why the actual VCR was not included in the system and a VCR program was constructed to simulate the physical device. Given that we can find a VCR with suitable feedback capabilities, it can be used to replace the VCR program in the future.

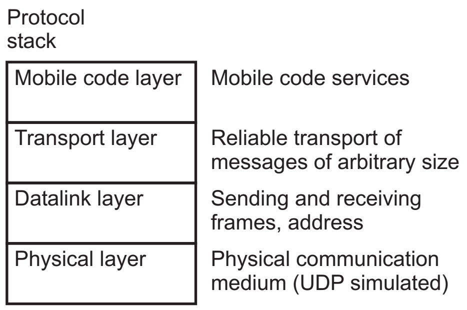
Protocol stack
Mobile code layer
Mobile code services
Transport layer
Reliable transport of messages of arbitrary size
Datalink layer
Sending and receiving frames, address
Physical layer
Physical communication medium (UDP simulated)

Figure 3. Communication protocol stack for the example system.

Communication channel

The VCR adapter and the VCR program were run in one computer and the client was run in another. In the example system, these computers were connected to the network by using the TCP/IP protocol. We constructed a simple communication protocol stack and used it for communication between the two computers. The stack is presented in Figure 3. The stack operates on top of a TCP/IP protocol stack and follows loosely the OSI reference model [15]. The stack consists of four layers: a physical layer, a datalink layer, a transport layer and a mobile code layer.

The physical layer was simulated using the UDP broadcast packets provided by the TCP/IP protocol. This was done to model the operation of communication devices that use radio waves or some other broadcast medium for communication. In broadcast communication, each data frame is received by every active system within the communication range.

The datalink layer provides services for sending and receiving directed and broadcast frames. Directed frames contain both source and destination addresses and are received only by the host indicated by the destination address. Broadcast frames contain only the source address and are received by all the active systems, except the broadcasting system.

The transport layer provides services for reliably transmitting and receiving messages of arbitrary length. It also provides a service for locating other active systems.

The mobile code layer provides the following services:

- Locating active systems
- Requesting information from the selected system
- Requesting a mobile code of the desired type from the selected system
- Transmitting the mobile code with the initialization parameters to the selected system for execution
- Transmitting data messages of arbitrary length to the selected system

The mobile code protocol is a very lightweight message-based protocol. The reason for this was the desire to keep the embedded system as simple as possible. Our goal was to provide only basic functionality that could be used to implement more complex communication mechanism, such as the Remote Procedure Call (RPC) system, if that were required in some applications.

Data messages of arbitrary length are sent through the mobile code layer using the notion of channels. When a data message is sent, an integer number indicating the channel is attached to it. The receiving mobile code layer can use this channel number to identify the destination component. In a sense, the channel number is similar to the port number in the TCP/IP protocol.

The protocol stack described above requires only a physical device with a capacity to broadcast fixed-length data frames to be utilized in various applications. Furthermore, the use of existing protocol stacks with the mobile code layer is very straightforward: it only requires services that can be used to locate and reliably transfer messages of an arbitrary length.

The main reason for constructing a complete ad-hoc protocol stack was the desire to model the operation of a low-range packet-based radio network as well as possible without actually using any wireless communication devices. We wanted to test the system first in a familiar environment, which could be easily monitored. We will use a standard protocol stack to replace the layers below the mobile code protocol if a suitable one can be found when, for example, Bluetooth is released.

Client

The purpose of the client system was to simulate a device that could be used to display virtual user interfaces

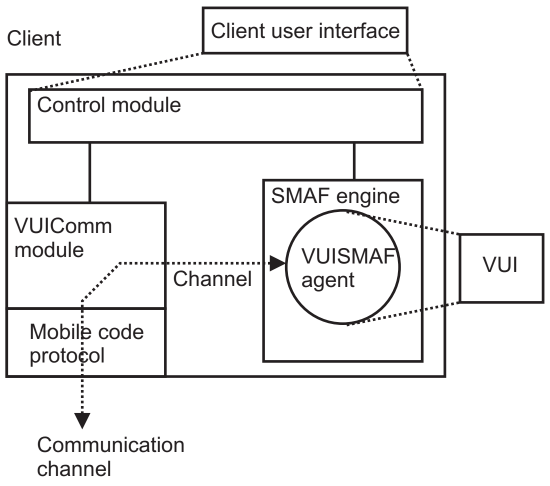

Figure 4. Structure of the client system.

for different embedded systems. Such devices include PDAs, future cellular phones and laptop computers. The structure of the client system is presented in Figure 4. The client system consists of the following software components:

- Mobile code protocol
- Control module
- Modified SMAF engine
- VUIComm module

The mobile code protocol operates as described earlier in this Chapter. The control module coordinates the co-operation of the other software modules. In addition, it also creates a user interface for the client. The user utilizes this user interface to control the client.

The Simple Mobile Agent Facility (SMAF) system (also referred as the SMAF engine) is used to run the VUI code. The SMAF system is a mobile code framework constructed by the authors to operate as a test bench for various mobile code paradigms. It also provides more profound knowledge of the implementation techniques needed for the mobile code systems. The SMAF system was not intended to compete in functionality with any of the existing mobile code frameworks.

The SMAF system was constructed using the SUN JDK (Java Development Kit) version 1.2. Java was selected as the implementation language for several reasons. First of all, Java contains several features that enable relatively easy implementation of various mobile code paradigms. Secondly, Java is platform independent in the sense that the compiled Java byte code is interpreted by the Java virtual machine (JVM). JVM is currently available for all the major computer platforms.

The overall architecture of the SMAF system follows the guidelines of the Mobile Agent System Interoperability Facility (MASIF) specification [16] from the Object Management Group (OMG). MASIF was formerly known as the Mobile Agent Facility (MAF) specification. The SMAF system operates as an execution environment for SMAF agents by providing basic services, such as agent transfer from one SMAF system to another.

When a SMAF agent is created, the Java byte codes that make up the agent are stored in an archive congruent with the Java Archive (JAR) file format. These byte codes are used for subsequent class loading during the agent's lifetime. The JAR file is transferred with the agent execution state when the agent moves from one SMAF system to another. In this way the agent does not need any external code servers for class loading during its lifetime. When implementing mobile code paradigms, the SMAF system utilizes a weakly mobile technology [14]. Systems utilizing a weakly mobile code technology do not preserve the precise execution state of the mobile code when the code is transferred from one system to another. Instead, only parts of the execution state are stored at the source host and transferred with the mobile code to the destination. The receiver uses this data to reconstruct some parts of the mobile code execution state. If the execution needs to be resumed, it can be implemented by, for example, using state variables to select the desired execution branch at the receiver.

The weakly mobile technology is implemented in the SMAF system by using Java Object Serialization. The mobile code state is serialized and the code is transferred to the destination with the serialized state data. When the mobile code is reconstructed at the receiver, the serialized state data is used only to restore the values of the objects' member variables. The state of local the variables in the member functions is lost.

The SMAF system used in the example system was modified slightly, so that it could be used to execute special virtual user interface agents (VUISMAF agent). The VUISMAF agent differs from the ordinary SMAF agent only in the sense that it contains additional functions that can be used to communicate with the embedded system where the agent code originated. The client in the example system can only execute a mobile code that is congruent with the VUISMAF or the SMAF agent structure.

The VUIComm module acts as a relay between the control module and the module that implements the mobile code protocol. It does this by creating and maintaining a channel list that is used to direct data received from the mobile code protocol to the correct virtual user interface. When a new VUI is created, the VUIComm module creates and assigns a channel object to that VUI. This channel object can be used to send and receive messages between the VCR adapter and the VUI. In other words, by using the channel objects, the virtual user interface can remain unaware of the actual mechanism that is used for communication between the client and the embedded system. The current version of the VUIComm module is implemented by assuming that VUISMAF agents will not move from one SMAF system to another, although this is possible.

VCR adapter

The purpose of the VCR adapter is to simulate a device that could be used to implement the virtual user interface functionality for different embedded systems. The VCR adapter acts as an interface between the actual VCR device and the rest of the system. The structure of the VCR adapter is presented in Figure 5.

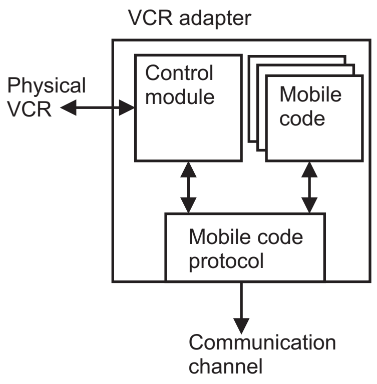

Figure 5. The structure of the VCR adapter.

As shown in the Figure, the VCR adapter consists of three components:

- Mobile code protocol
- Control module
- Mobile code

The mobile code protocol was described earlier in this Chapter.

The control module contains detailed information about the actual VCR device. It receives commands from the virtual user interface through the mobile code protocol and converts these commands into physical signals that manipulate the VCR. In addition, the control module transmits the messages that reflect the status of the VCR device to the virtual user interface. The mobile code contained in the VCR adapter follows the VUIS-MAF agent structure.

Virtual user interface

Figure 6 presents the virtual user interface for the example VCR device. It resembles a user interface for a standard VCR. The mobile code creates this VUI by using the standard Java class libraries located at the client. In other words, the mobile code uses the user interface primitives provided by the standard Java class library to construct a desired user interface.

When the user presses the control buttons, such as Play and Stop, the VUI sends command messages to the VCR adapter. These command messages have a very simple structure, consisting of only a few bytes. The VCR adapter receives these command messages and transforms them into internal messages. These internal messages are then sent to the program simulating the actual VCR device. The VCR adapter constructs status messages from the replies it receives from the VCR device and sends them to every virtual user interface that is currently active. In this way a given VCR can be inspected and manipulated by several virtual user interfaces at the same time.

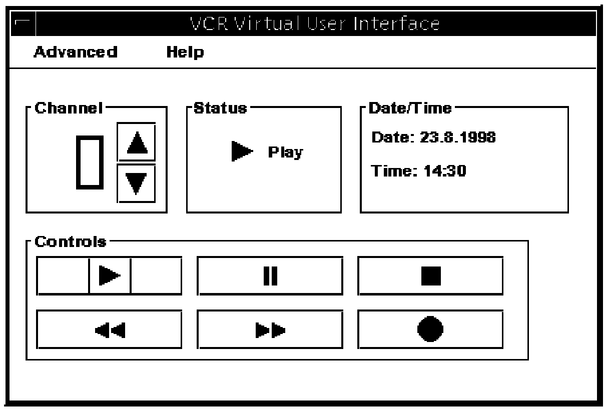

Figure 6. Virtual user interface for the VCR device.

The size of the JAR file containing the mobile code for the virtual user interface presented in Figure 7 is approximately 25 Kbytes. However, the current version of the interface was not designed or programmed so that the amount of code would be minimized. It is probable that the size of the JAR file can be reduced to fewer than 20 Kbytes.

5. Discussion

The technique for creating virtual user interfaces by using a mobile code offers several benefits compared to the existing approaches described in Chapter 2, but it also has some drawbacks. The benefits and drawbacks must be weighed carefully for every application if this technique is to be utilized in an actual system. This Chapter explains the benefits and drawbacks to allow proper evaluation of the suitability of this technique.

Benefits

The mobile code VUIs offer several benefits when compared to the existing approaches. The mobile code approach:

- is flexible, simple, and easy to implement
- is adaptable and utilizes efficient local communication
- is open
- allows a diverse set of value-added services

Flexibility is one of the main benefits of the mobile code VUIs. The mobile code uses the services provided by the execution engine to implement a desired user interface. Even though different execution engines and client systems may impose restrictions on the behavior of the mobile code, the approach presented in this paper does not in itself restrict functionality in any way.

Furthermore, flexibility does not make the approach complex or hard to implement. Traditional programming conventions and languages can be used to program the software for the embedded system, and productive high-level programming languages, such as Java, can simultaneously be used for the VUI. The internal communication protocol between the embedded system and the VUI can also be tailored to suit the needs of each particular application.

Even an existing embedded system would require only minor and inexpensive modifications to include VUI functionality. The easiest way to include a VUI in an existing embedded system would be to add an I/O port for an external adapter, as in the VCR example. This port could be used to receive commands from the adapter and to send out status messages reflecting the status of the embedded system. The adapter would provide the VUI functionality, and the embedded system itself would need no further modifications. This adapter could be sold as an option for each embedded system.

Adaptation is one of the most important benefits when the mobile code approach is used in mobile computing. The mobile code VUIs allow mobile users to control diverse embedded systems without knowing anything specific about them. The only requirement is that the client and the embedded system can communicate with each other and that the client can execute the VUI code. It is a lot easier to agree on standard interfaces for execution engines than for a diverse set of embedded systems, at least if the execution engine provides sufficient services for the mobile code. Adaptation would be hard without a mobile code. The client would have to share a control protocol with every embedded system, or to have a separate control program installed for every possible system.

Efficient local communication is another important benefit for mobile computing. Embedded systems can be controlled by using a local communication, which means that low-power, high-bandwidth communication devices can be used and the client is not required to maintain a possibly expensive connection with the Internet or a remote server. However, flexibility ensures that the VUI can obtain some parts of the mobile code from the remote servers, possibly by using the network connection provided by the client. In this way, the infrequently used parts of the mobile code, or possibly the parts that implement functionality that requires network connection, can be obtained from the remote server.

The mobile code VUIs also offer an open solution, as the approach does not require any special programming language or hardware to be used. An embedded system may even contain a different mobile code to be used with different clients. Furthermore, if the mobile code in the embedded system is stored in a re-programmable ROM memory, services facilitating software updates can be easily implemented. The main reason for easy update is the clear distinction between the mobile code and the control software in the embedded system. The mobile code is not executed by the embedded system and dynamic software update techniques are therefore not required.

As an example, consider a situation where a new type of client with a new mobile code engine is introduced. If the user wants to use this client instead of the old one, he or she simply obtains a new VUI mobile code and updates it in the embedded system. One way to do this is to include a special update code in the embedded system that can be used to create a special VUI for software updates. This VUI code can then obtain the new mobile code from the remote server and upload it to the embedded system. Software updates can naturally be done if the mobile code is stored in a separate ROM chip that can be replaced with a new one. In this way, the software in the embedded system can be kept simpler, as it does not need to provide software update services.

The most interesting advantage in the mobile code approach is how different types of value-added services, which have previously been very hard to implement, if not totally unfeasible, could be provided for embedded systems. One example is the software update service described above. Another example is the integration of embedded systems in external services. For example, the VCR VUI can use an Internet connection provided by the client to contact a remote TV guide server in order to obtain TV program data. The user could then use this data to select the programs to be recorded and the VUI would convert the selections to record commands and sent them to the VCR. The mobile code for this service could be stored in a server in the Internet, so that it could be easily modified if there were a change, for example, in the TV guide server. Storing the mobile code in a remote server makes sense in this system, as this service requires an Internet connection for obtaining the TV guide data.

Drawbacks

The approach presented in this paper does not come without drawbacks. These drawbacks include:

- A need for additional hardware and maintenance
- Communication overhead in some situations
- Code mobility is still a relatively immature technology
- Information security

As described in Chapter 3, the main requirement for the mobile code approach is that the embedded system contains communication hardware. Furthermore, storing the mobile code in the embedded system requires a variable amount of non-volatile memory. The addition of this extra hardware can be problematic in systems where the total cost of the product must be kept to the absolute minimum and the virtual user interface does not result in substantial competitive advantage. However, some form of communication hardware is required in any case if we want to add remote control functionality to an embedded system.

Maintenance may also be a problem when the mobile code is stored in the embedded system. For example, non-automated software updates are completely unsuitable if embedded systems become truly ubiquitous. We hope that most of the software updating can be automated, as outlined earlier in this Chapter.

Downloading the mobile code from the embedded system can also increase the communication overhead in situations where the actual communication session only lasts for a short period. This can be reduced or, in some situations, prevented by using mobile code caching at the client. For example, the frequently used mobile code, such as that used to control embedded systems at home, can remain stored in a client's cache.

The code mobility, while not new as a concept, is still relatively immature and not applied in a large scale. The designers and developers lack experience compared to their colleagues, who are using more traditional programming paradigms, such as the client-server. This can be very problematic, especially as it may be difficult to provide acceptable security in mobile computing.

Information security is probably the most important problem in mobile code VUIs that needs to be solved. Even though it is not recommended, we may ignore most of the security problems if the mobile code approach is used only at home or in other relatively closed environments, much like security is mostly ignored in the current remote control units. For example, we may be tempted to assume that we never encounter malicious mobile code or that no eavesdropping is possible.

However, information security is an essential requirement in mobile computing. In mobile computing, the user may want or need to control unknown systems, which makes it necessary to be able to shield clients against malicious mobile code. In addition, we cannot ignore eavesdropping if the communication involves sensitive data. Furthermore, embedded systems must also be shielded against unauthorized use.

The techniques providing information security for mobile computing, especially for systems utilizing mobile code, are still very immature. Most of the security techniques for mobile code systems have been designed for applications with specified features and restrictions, and are not necessarily suitable for other applications. In a sense, however, the security requirements for mobile code VUIs are very similar to the security requirements for Java applets, and we are therefore optimistic and expect that a suitable security mechanism can be constructed. Because security in mobile code systems in general is a very complex issue, only a brief outline of the problems and solutions is presented here. For a more complete description of the problem and the possible solutions, refer to [17]. The security requirements include:

- Protecting the client against malicious mobile code
- Authenticating the client (or the user)
- Authenticating the embedded system
- Preventing eavesdropping
- Ensuring mobile code integrity
- Protecting the executing mobile code against malicious clients

The client should be shielded in such a way that the downloaded mobile code is not able to perform malicious actions once executed. This can be implemented by a sandbox security model that is used to limit the operations that can be performed by the mobile code. An example of such sandbox security can be seen in [18].

The authentication of both communicating parties can be performed by using digital signatures. Authentication is important in systems that need to restrict the number of persons allowed to control the system. Furthermore, if the source of the mobile code can be authenticated, varying sandbox restrictions can be utilized on the mobile code, depending on the degree of trust the client has in that particular source.

Communication can be encrypted to prevent eavesdropping [19]. Eavesdropping must be prevented in systems that transmit sensitive information that should not be exposed to outsiders.

Digital signatures can also be used to ensure mobile code integrity. Mobile code integrity ensures that the code is not altered after its distribution.

Protecting the mobile code against malicious clients is a very hard problem, mainly because the client can inspect the code during its execution. Initial solutions on how to protect the mobile code from inspection suggest that the mobile code should be constructed so that the actual purpose of execution is not revealed. Code obfuscation [20] or cryptographic solutions [21] can be used to achieve this. Malicious clients may, for example, inspect mobile code to find ways to control the embedded system without authorization.

As mentioned earlier, information security in mobile code systems is a very complex issue and largely dependent of the different applications at hand. The level of security acceptable for a VCR device with a short-range IR transceiver may be totally unacceptable for some other system, such as the access control systems of our future homes.

6. Conclusion and Future Work

This paper described a flexible, yet powerful concept of a mobile code used to control ubiquitous and diverse embedded systems. The approach to create mobile code VUIs offers several benefits compared to the alternative techniques. These benefits include simplicity, efficient local communication, adaptation, openness, and most notably, flexibility. Furthermore, the fact that a single simple device can be used to control different embedded systems offers a cost-efficient and convenient control solution. The approach is well suited for the highly dynamic computing environments of mobile computing in the future. However, it is also usable in static environments.

In a wider sense, our goal is to inspect the possibility to use mobile code as an interconnecting glue to integrate various embedded systems. The mobile code VUI concept presented in this paper is the first step towards this goal. We feel that the concept can be utilized and extended in future embedded systems to provide other functionality than just control. Such functionality could include delivering communication protocols and device drivers as mobile code to the other devices in the system. It is also interesting to envision the range of possible value-added services that could easily be implemented for different systems.

However, further work is needed to construct actual clients and embedded systems that utilize the mobile code concept in a way that allows it to be tested in a real-world situation. We have already constructed an early version of a system that uses radio modems for communication. In addition, we have constructed several early prototypes of virtual user interfaces for various embedded systems and also for other real-world objects. These prototype VUIs will be utilized in our future research as we try to fully understand the potential offered by the mobile code. We are also actively looking for suitable security techniques for VUIs with varying security requirements.

7. References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, September 1991, pp. 66-75.

[2] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen, and W. Allen, "Bluetooth: Visions, Goals, and Architecture," *Mobile Computing and Communications Review*, Vol. 2, No. 4, 1998, pp. 38-45.

[3] M. Schlett, "Trends in Embedded Microprocessor Design," *Computer*, August 1998, pp. 44-49.

[4] V. V. Badami and N. W. Chbat, "Home Appliances Get Smart," *IEEE Spectrum*, August 1998, pp. 36-43.

[5] K. R. Wood, T. Richardson, F. Bennet, A. Harter, and A. Hopper, "Global Teleporting with Java: Toward Ubiquitous Personalized Computing," *Computer*, February 1997, pp. 53-59.

[6] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, January-February 1998, pp. 33-38.

[7] I. D. Agranat, "Engineering Web Technologies For Embedded Applications," *IEEE Internet Computing*, May-June 1998, pp. 40-45.

[8] R. Itschner, C. Pommerell, and M. Rutishauser, "GLASS: Remote Monitoring of Embedded Systems In Power Engineering," *IEEE Internet Computing*, May-June 1998, pp. 46-52.

[9] A. Puliafito, O. Tomarchio, L. Vita, and K. S. Trivedi, "Increasing Application Accessibility Through Java," *IEEE Internet Computing*, July-August 1998, pp. 70-77.

[10] S. M. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. W. Tickey, and P. Winterbottom, "The Inferno Operating System," *Bell Labs Technical Journal*, Vol. 2, No 1, 1997, pp. 5-18.

[11] Sun Microsystems, Inc., "Jini Technology," 1998, http://java.sun.com/products/jini/

[12] T. D. Hodes, R. H. Katz, E. Servan-Schreiber, and L. Rowe, "Composable Ad-hoc Mobile Services for Universal Interaction," *Proc. of the 3rd annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM'97)*, September 26-30, 1997, Budapest, Hungary, pp. 1-12.

[13] G. Kortuem, Z. Segall, and M. Bauer, "Context-Aware, Adaptive Wearable Computers as Remote Interfaces to 'Intelligent' Environments," *Proc. of the 2nd Int. Symp. on Wearable Computers (ISWC'98)*, October 19-20, 1998, Pittsburgh, Pennsylvania, USA.

[14] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, May 1998, pp. 342-361.

[15] M. Rose, "The Open Book: A Practical Perspective on OSI," Prentice-Hall, Inc, New Jersey, 1990, pp. 651.

[16] Object Managegent Group (OMG), "Mobile Agents Revision," July 28, 1998, (MASIF Revision), Available online ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf

[17] D. Chess, "Security Issues in Mobile Code Systems," G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 1-14.

[18] G. Karjoth, D. Lange, and M. Oshima, “A Security Model for Aglets,” *IEEE Internet Computing*, July-August 1997, pp. 68-77.

[19] C. Kaufman, R. Perlman, and M. Speciner, “Network Security: Private Communication in a Public World,” Prentice Hall PTR, New Jersey, 1995. pp. 505.

[20] F. Hohl, “Time Limited Blackbox Security: Protecting Mobile Code Agents From Malicious Hosts,” G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 92-113.

[21] T. Sander and C. Tschudin. “Protecting Mobile Agents Against Malicious Hosts” G. Vigna (Ed.) Mobile Agents and Security LNCS 1419, Springer-Verlag, Berlin, 1998, pp. 44-60.