Palmtop Programmable Appliance Controls *†

Diomidis Spinellis
University of the Aegean
dspin@aegean.gr

Abstract

Palmtop PCs with infrared transceivers provide a user-friendly, intelligent, and extensible alternative to the remote controls traditionally used to control home appliances. We describe the design and implementation of a palmtop programmable appliance control system. The system is designed around RDL, a domain specific language, allowing the realisation of virtual remote control units and sophisticated interaction sequences. The multitude and diversity of control applications programmed in RDL point towards a new appliance control paradigm based on a client-server architecture and intelligent user interfaces.

Keywords: *Remote control; infrared interfacing; domain specific language.*

1 Introduction

We are constantly witnessing the rapid convergence of many consumer and information technology appliances. An interesting application of this convergence that has received relatively less attention than the widely publicised convergence of the World Wide Web and television is the remote control of consumer appliances using personal computers. The universal adoption of the remote control as the primary appliance interface combined with the miniaturisation of personal computers to the size of palmtop devices has made possible the realisation of truly programmable, versatile, and user friendly appliance control systems.

During the last five years we have experimented with a number of such technologies and implemented these designs on the increasingly powerful palmtop devices that appeared on the market. This paper describes the design, implementation, and use of a palmtop-based universal appliance control that can be programmed using a lightweight domain-specific language. The system we describe is publicly available [1] and has been used in a number of diverse applications. The remainder of this paper is structured as follows: in section 2 we briefly describe the salient technological characteristics related to appliance remote controls, palmtop computer infrared interfacing, and domain-specific languages; section 3 provides the design of the remote control system; section 4 contains important implementation details; section 5 outlines some interesting applications of our system as reported by other users. Section 6 concludes the paper by providing pointers to possible future enhancements and technological advances.

2 Technology Overview

The system we describe in this paper bridges a set of three diverse technologies: infrared remote controls, infrared interfacing of palmtop computers, and domain-specific languages.

2.1 Remote Controls

Historically consumer appliance remote controls were developed to provide a convenient way to control the appliance without having to move near it. Early remote controls were essentially detachable control panels connected to the appliance through a thick umbilical cord. The introduction of solid state electronics and integrated circuits allowed the development of battery-powered wireless remote controls. Although radio frequency and ultrasound technologies have been used in the past to provide the communication path between the control and the appliance, nowadays virtually all appliance remote controls are based on optoelectronic components communicating using infrared light signals. The evolution of appliance controls currently moves towards the transfer of functionality from the appliance to the remote control. A number of modern appliances provide only minimal or no control on the main appliance and delegate all functionality to the remote control. Notable such examples are VCRs, televisions, and air-conditioners. Another interesting and helpful for our application trend is the inclusion of remote control interfaces on appliances where such a

Personal Technologies, 2(1):11–17, March 1998.

†This is a machine-readable rendering of a working paper draft that led to a publication. The publication should always be cited in preference to this draft using the reference in the previous footnote. This material is presented to ensure timely dissemination of scholarly and technical work.

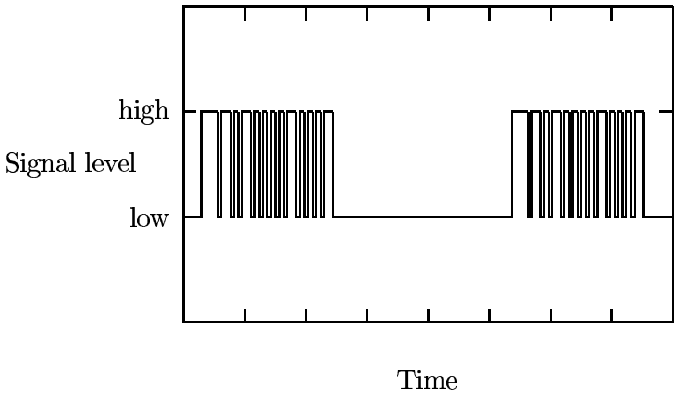

Figure 1: Sample of a 12-bit remote control signal.

control might seem illogical such as car radios and personal stereos.

Most remote controls provide essentially one-directional links between the control and the appliance. The reverse feedback of the control loop is usually provided directly to the human in the form of an indicator on the appliance, or altered appliance behaviour. Contrary to what a rational technologist might expect, the transmission format of most remote controls, although similar at the lowest (physical link) layer, diverges phenomenally when one examines the encoding method used to transmit the command. It is this variety of encoding methods that ensures that remote controls from different manufacturers do not control other appliances (they still interfere with each other though and therefore can not be used at the same time). The majority of remote controls encode commands by pulse modulating a 980nm infrared light signal at a carrier frequency ranging from 32KHz to 40KHz.

The modulated sequence typically transmits a binary word using a self-clocking mechanism. An example of such a sequence is shown in Figure 1. Word length varies from 12 to 48 bits. At the beginning of each word a continuous signal burst is often used as a mechanism to tune the receiver's automatic gain control circuit. Bits are encoded using either pulse width or biphase encoding. More complicated schemes such as clock-based serial transmission are often used for communication between more intelligent devices such as computers with infrared ports.

A number of industry standards such as RC-5 and RC-6 by Philips partition the code space into areas representing the manufacturer, the application (TV, VCR, CD, etc.), and the function. Most code spaces are sparsely populated. Interesting undocumented functions are often hidden in the code space. As an example, by a sequential search through the code space we have discovered:

- the possibility to "mute" the volume of a MIDI system that did not provide such a function on the remote control,
- to provide direct access to CD seek functionality on a portable CD player that only provided such functionality using a mode change control on the front panel, and
- to sequentially "zap" between all radio stations on a radio tuner.

2.2 Infrared Interfacing

Palmtop computers are being increasingly used as platforms of choice for advanced human-computer interaction applications [2]. Although we performed the first infrared control experiments on desktop workstations using custom decoding circuits, we quickly decided to use as our platform the Hewlett-Packard palmtop PCs (HP-95LX, HP-100LX, HP-200LX). The infrared interfacing circuit is essentially the same on all three machines. The infrared transmitter consists of an infrared LED that can be driven in a variety of ways: in serial communications mode through the output of the UART (universal asynchronous receiver transmitter) chip, in software controlled mode by directly pulsing a given bit, or in modulated software controlled mode. The last way, which is the one we used, allows the generation of modulated signals by using the UART chip 1.84MHz baud rate generator clock as the modulation source and controlling it by switching a single bit.

Although most off-the-shelf remote control receiver modules (such as the Sharp GP1U28XP) provide integrated demodulation functionality, the general purpose infrared receiver hardware of the palmtop PC does not have that capability. It does however provide a latching "event" bit which is set to *one* whenever an infrared pulse is received. The bit is "sticky" and needs to be reset by software to the *zero* state. Using this bit, it is possible to construct a digital low-pass filter to provide the desired 40KHz demodulation function.

Lately, an industry body called the Infrared Data Association (IrDA) was formed aiming to create an interoperable, low cost, low power, half-duplex serial data interconnection standard to support a walk-up, point-to-point user model. Although the IrDA serial infrared physical layer link (IrDA-SIR) [3] specification is based on the same technology as our application, the modulation specified is not compatible with the formats used by remote controls and will therefore not concern us in this article.

2.3 Domain Specific Languages

A domain-specific language [4] is a programming language tailored specifically for an application domain: rather than being general purpose it captures precisely the domain's semantics. Examples of domain-specific languages include *yacc* used for program parsing, *html* used for document mark-up, and *vhdl* used for hardware descriptions. Domain-specific languages allow the concise description of an application's logic reducing the semantic distance between the problem and the program [5]. As a design choice for the programming interface of an appliance control a domain-specific language presents a number of distinct advantages over a "hard-coded" program logic or a graphical user interface:

Versatility A small set of language statements allows the specification of an unlimited set of remote control features and functions. Since the public release of our system we were informed of many applications we could not have envisaged during its design.

Small Footprint Although palmtop computers are becoming increasingly powerful, the memory requirements of an application still matter. A domain-specific language interpreter can be implemented under tight memory constraints. As an additional advantage, small applications start up relatively quickly; an important characteristic for a remote control.

User Control The end user can control most aspects of the application. The application's core engine can remain stable, allowing implementation effort to be directed in optimising its domain-related functionality, rather than adding gratuitous bells and whistles.

Concrete Expression of Domain Knowledge Domain-specific functionality is not coded into the system or stored in an arcane file format; it is captured in a concrete human-readable form. Programs expressed in the domain-specific language can be split, combined, shared, published, put under release control, printed, commented, and can even be automatically generated by other applications.

3 Design

The system architecture is split into three parts:

- a generic infrared interface engine,
- a domain-specific language interpreter, and
- an infrared command recording interface.

The domain-specific language, termed Remote Definition Language (RDL), provides commands to draw the screen, and key bindings to send the appropriate remote control codes. In the following paragraphs we provide a brief description of the RDL syntax and semantics.

3.1 Lexical Structure

An RDL program is stored as a single file. Different files can be used to organise control sets and user interfaces. An RDL file consists of a set of commands. Every command must start on a new line; this simplifies the design of the parser and the interpreter's implementation. White-space and blank lines are ignored. Lines that begin using the "#" character are regarded as comments. Remote control signals are relatively compact and are therefore represented as inline ASCII strings using a portable encoding convention.

3.2 Program Structure

The commands inside an RDL definition file are organised as procedure definitions. No commands are allowed outside the scope of a procedure definition. A procedure definition is structured as follows:

```
define <procname>
    <command>
    ...
end
```

This construct defines a procedure named *procname*. The commands are executed when the procedure is invoked using the command *call procname*.

3.3 Command Description

The minimalistic nature of RDL is exemplified by the nine commands it supports:

key '*character command*

key *number command* Installs a key binding for the specified key. When a key is pressed the command defined by the binding is executed. ASCII keys are specified using the 'character notation. Other keys, such as arrow keys, are specified using key's scan code in hexadecimal following the C language *0x* convention.

clkey Clears all key bindings.

send "*string***"** Sends the raw code signals specified by the string to the infrared control port.

call *procname* Upon execution of the call command control is transferred to the named procedure. When the procedure's *end* statement is reached control is transferred back to the command after the call command.

cls Clears the screen contents.

image *filename* Displays the image file specified by filename.

print *row column "string"* Prints a string at the row and column specified.

pause *seconds* Pauses the specified number of seconds.

exit Terminates the program.

3.4 Execution Semantics

Program execution starts from the procedure called *main*. Control proceeds serially from that point transferring control when a call command is encountered. When the called procedure has been executed (i.e. the end command has been reached) control is transferred back to the command immediately following the call command. As the language does not provide any imperative input statements the *main* procedure normally terminates after some initial processing. At that point the system enters an event loop which executes commands based on keyboard input events and the corresponding key definitions. The program terminates when the *exit* command is executed. A simple way to call procedures based on the current date and time is provided based on a dynamic procedure name substitution mechanism.

A minimal file for the control of a simple on/off device is the following:

```
# Minimal remote control file
define main
    print 0 0 "+):On -):Off ESC):Quit"
    key 0x1b exit
    key '+ send "##'##[##'#%+##'##X##'##X"
    key '- send "AAA:#[[['#%+##'ZcXcc'ccY"
end
```

A remote control file for controlling two devices with a mode change (e.g. a TV and a VCR) is illustrated in Figure 2. RDL programs can be used to create new features that are not available in the original products. As an example the RDL program in Figure 3 implements a "zap" command that cycles through three TV stations.

3.5 The Infrared Command Recording Interface

The system provides a mode used to record remote control commands. The recording process consists of pressing a remote control key in order to send the infrared signal to the palmtop PC and subsequently pressing the palmtop key that will correspond to the remote command. The result of this process is a complete working RDL program. The program consists of assignments of appropriate key bindings and *send* commands for all infrared sequences that

```
define main
    cls
    clkey
    print 0 0 "t) TV v) Video ESC) Quit"
    key 't call tv
    key 'v call video
    key 0x1b exit
end

define tv
    print 0 0 "ESC) Main"
    cls
    clkey
    key 0x1b call main
    # TV command key bindings follow here
end

define video
    ...
```

Figure 2: RDL example of a mode change.

```
define z1
    send "##'##[##'#%+##'##X##'##Y"
    key 'z call z2
end

define z2
    send "##'##[##'#%+##'##A##'##B"
    key 'z call z3
end

define z3
    send "##'##[##'#%+##'##M##'##N"
    key 'z call z1
end

main
    key 'z call z1
    ...
```

Figure 3: RDL example of a *zap* button implementation.

were recorded. This RDL program can either be used directly, or it can be customised by adding user interface elements, merging it with other RDL programs, splitting the key bindings into different screens, or enhancing it in other creative ways. The generation of a working RDL program from the recording process makes the system immediately useful, without requiring its users to learn any RDL syntax.

4 Implementation

Similar to the application design, the implementation is split into two parts:

- the infrared transceiver engine, and
- the RDL interpreter.

The infrared transceiver engine has to remove the 40KHz carrier signal from remote signals being recorded and store the resulting waveform in a compact form. When a remote control signal is to be transmitted it has to modulate the stored waveform using a 40KHz carrier and send back the same waveform. The 40KHz carrier signal is removed using a digital bandpass filter based on the latching feature of the infrared receiver circuit. Incoming infrared signals set a bit which has to be cleared by software. The receiver recorder loops while a signal is received testing the bit and immediately setting it to 0. Since the 40KHz modulation period (25 μs) is less than half of the period of a complete recorder loop, the loop will always find the bit set to 1 as long as a carrier frequency is being transmitted. The modulation of the waveform during transmission is a lot simpler. The transmitter circuit can modulate the outgoing signal using the output of the serial port BAUD-rate generator. By setting the divisor registers to the appropriate value, the BAUD-rate generator 1.84MHz clock can be used to derive a 40KHz modulation frequency.

A second implementation difficulty of the infrared transceiver was the reliable recording and playback of remote control waveforms. Since the palmtop PC clock circuits do not provide sufficient granularity to determine the length of the infrared waveform pulses the transceiver is based on software-controlled counting loops. A major problem is the timing synchronisation of waveform recording and playback. This is solved by factoring the receiving and transmission functions into the same loop thus ensuring that the code execution paths are similar. Compiler-assigned register variables are common to both parts; with a careful choice of code statements the receiving loop takes the same number of clock cycles as the transmission loop. These techniques allowed us to code the transceiver in a high-level language (C), insulate the hardware dependencies, and painlessly integrate the transceiver with the rest of the system.

Figure 4: A virtual remote control screen.

An important implementation objective of the RDL interpreter was a small memory footprint; an essential feature given the limited memory of palmtop PCs, and the additional memory requirements of the palmtop's application shell. The interpreter is implemented using a simple recursive-descent parser [6, p. 181] and a hand-crafted lexical analyser. The RDL file is never tokenised or read into memory as is the common case in most interpreted languages. Instead, capitalising on the speed of the palmtop memory-based filesystem, interpretation is performed directly on the file contents. In order to increase the application's responsiveness — memory-based file access still suffers from the operating system overheads — an aggressive caching strategy is implemented. The file offsets of the most recently used procedure definitions are stored in an in-memory structure allowing fast random access to their contents. More importantly, all key bindings are only stored as file offsets to the actual code sequence that they have been bound to.

5 Interesting Applications

The system has been publicly available over the Internet for over four years. During that period we received positive feedback from many users and became aware of applications that we had not envisaged during its initial design. The most obvious use of the system is the implementation of a *master remote control*. A number of virtual remote controls are unified as a single palmtop-based unit. Users can typically select the device they want to address and the palmtop's screen changes to reflect the user-interface for

the specific control. Most users try not to mimic the functionality of the controls they are replacing, but to provide a unified user-interface for all appliances. This user-interface re-engineering often produces interfaces that are more user-friendly than the original one. This should come as no surprise given the abysmal quality of some electronic appliance user-interfaces [7, p. 174–177]. The full keyboard of the palmtop can also be used to simplify the programming of devices that rely on complicated key sequences for their programming. As an example a user has implemented a mini-disk title editor and database [8], that uses the palmtop's alphanumeric keyboard instead of the more limited keyboard provided by the mini-disk remote control. The ability to start-up a palmtop application at a specific time allows the timed control of appliances such as air-conditioning units and the simplified programming of VCRs. The repetition of palmtop timing events can be flexibly programmed allowing an infinite variety of applications.

Unfortunately, in addition to a number of original and unanticipated uses of our system, we also became aware of a more sinister application domain. Some security systems used in cars and homes are remote controlled using infrared signals. Many controls use a fixed 4–32 bit signal to open a car's doors or deactivate the alarm system security. This signal can be easily intercepted and recorded by a criminal and played back at a latter time in order to deactivate the security system. The problem lies in the design of the security systems which use a immutable control signal instead of a cryptographically secure authentication protocol. The problem is not unique to infrared-controlled security systems; unencrypted passwords are used by most Internet protocols (FTP, TELNET, POP3). In both cases advances in technology and the sophistication of criminals have made it imperative to deploy secure authentication protocols. The computational power of a palmtop PC is the minimum required for experimenting with protocols based on public key encryption methods.

6 Conclusions

The design and implementation of an intelligent remote appliance control based on a domain specific language opened our mind to many interesting possibilities in the field of remote appliance control. RDL although capable of expressing a complicated remote control application lacks portability among appliances. It will be interesting to split it between an appliance-specific part implementing low-level functions (e.g. "fast forward", "rewind"), and a user-interface part . This architecture allows the distribution of appliance "drivers" and portable, sophisticated [9] user-interface programs. The portability of appliance user-interface programs and the ubiquity of appliance drivers can radically change the way we interact with appliances. Appliance manufacturers can distribute just the driver for their appliance as computer hardware manufactures do today. End-users will be free to choose the best universal appliance control available instead of stocking up remote controls of incompatible user interaction designs. Until this change materialises RDL programs and libraries running on palmtop PCs can act as middleware between the appliance control engine and the user-interface program.

The increased processing power, memory, and programmability offered by the proposed architecture at the remote control end opens up a number of interesting possibilities. Palmtop-based appliance controls can offer value-added services which the computationally and communication-challenged appliances can never hope to provide. Examples of these services are:

- the automatic programming of all appliance clocks with the correct time,
- the bootstrapping of tuners and TV sets with the user's preferred set of stations,
- the programming of air-conditioning units according to the season and the time of day,
- the maintenance of a consistent set of dial memories among the home, office, and the cellular phone, and
- the support of modern authentication protocols for security systems.

Eventually a client-server architecture [10] will evolve where appliances will just perform their basic function — record a TV broadcast on tape, wash the dishes, play a CD — and a single versatile, user-friendly, intelligent appliance control based on Windows CE or *Java Everywhere* will be the user's interface client.

Let the intelligent appliance control age begin.

References

[1] Diomidis Spinellis. Remote control 100. Online. ftp://eddie.mit.edu/pub/hp95lx/hp100lx/rc.zip, 1994. March 1998.

[2] Haruo Noma, Tsutomu Miyasato, and Fumio Kishino. A palmtop display for dextrous manipulation with haptic sensation. In *CHI '96. Conference Proceedings on Human Factors in Computing Systems*, pages 126–133. ACM, 1996.

[3] Joe Tajnai. *Serial Infrared Physical Layer Link Specification*. Infrared Data Association, version 1.2 edition, November 1997. Online. http://www.irda.org/standards/pubs/IrPHY_1_2.PDF. 11 April 1998.

[4] USENIX. *USENIX Conference on Domain-Specific Languages*, Santa Monica, CA, USA, October 1997.

[5] Diomidis Spinellis and V. Guruprasad. Lightweight languages as software engineering tools. In *USENIX Conference on Domain-Specific Languages*, pages 67–76, Santa Monica, CA, USA, October 1997. USENIX.

[6] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, 1985.

[7] Donald A. Norman. *The Psychology of Evereday Things*. BasicBooks, New York, NY, USA, 1988.

[8] Helmut Lucke. Using the HP200LX to label mini discs. Online. http://www.jyu.fi/minidisc/minidisc/-mdlbl/mdlbl.htm. 11 April 1998, August 1997.

[9] Thomas Baudel and Michel Beaudouin-Lafon. Charade: Remote control of objects using free-hand gestures. *Communications of the ACM*, 36(7):28–35, July 1993.

[10] Alok Sinha. Client-server computing. *Communications of the ACM*, 35(7):77–98, July 1992.