

End-User Programming in the Networked Home

Rob Hague, Alan F. Blackwell and Peter Robinson

University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue
Cambridge CB3 0FD
England, United Kingdom
{rob.hague, alan.blackwell, peter.robinson}@cl.cam.ac.uk

Abstract. Pervasive networking of domestic appliances provides a wealth of possibilities. Some of these possibilities will be anticipated by developers, but many will be novel and unexpected. Hence, the provision of end-user programming adds significant utility to a networked home. The work described investigates the design of end-user programming systems for a diverse user population, in the context of the Cambridge AutoHAN project.

Keywords. End-user programming, networked home, ubiquitous computing, tangible interfaces.

Paper presented at The 1st Equator IRC *Workshop on Ubiquitous Computing in Domestic Environments*, The School of Computer Science and Information Technology, The University of Nottingham, 13-14th September 2001. <http://www.equator.ac.uk/>

Introduction

Computers have become increasingly integrated into everyday life. This is most evident in an office environment, where the paraphernalia conventionally associated with computers - keyboards, monitors and mice - pervade the environment. However, it is also true, although less obvious, in many other environments, with the advent of increased automation of a wide variety of tasks. Bar code readers, speed cameras and mobile phones all contain significant computing power. The home environment is particularly rich in terms of computation; a typical home may have dozens of devices, each with its own microprocessor. In most cases, all but the most sophisticated devices communicate with each other in only the most perfunctory way. Integrating these disparate devices to provide a coherent computing environment is an interesting research problem, encompassing many aspects; networking, high-reliability computing, security and human-computer interaction. This work investigates aspects of the latter area, in particular the possibilities for end-user programming in such an environment.

Historically, the users of computers were programmers. As the field developed, ready-made applications became increasingly common, and the need for users to program decreased. This trend continued, spurred on by developments in user interface technology such as the mouse (Engelbart and English 1968), direct manipulation (Sutherland 1963), and the desktop metaphor, giving rise to the current situation where the vast majority of computer users use nothing but ready-made applications and have no knowledge of programming. The users of an application are referred to as “end users”, to distinguish them from the application developers, or users who have direct contact with, and possibly influence over, those developers.

While this has undeniably given a far wider range of people access to computing facilities, and resulted in computers being used in application areas as diverse as desktop publishing and air traffic control, it has also served to limit the flexibility with which those computing facilities may be used. End user applications are, by and large, generic, and users apply these applications to their specific problems. This “leaves personal computer users in an ironic situation,” where they are performing repetitive actions to apply the generic application to the task at hand, despite the truism that “computers are good at performing repetitive activities” (Cypher 1993). Without access to end user programming, the only solution is to request

additional features more applicable to your problem, or, more commonly, to await the release of the next version of the application in the hope that it will contain such features.

The size and diversity of the end-user population can only increase with the advent of “ubiquitous computing”, where computers become an invisible, integral part of the environment, much as electric motors did during the early twentieth century (Weiser 1991). While much research in ubiquitous computing has been centered in the business arena, both in and out of the office, another important area of research is ubiquitous computing in the home. A basic tenet of this area is that domestic devices are augmented with a network interface, allowing them to communicate with, and be controlled by, other devices. The key difference between this situation and other ubiquitous computing projects is the population of users. While a business will almost invariably employ a trained professional to set up and maintain any computing system, a home network may be set up by an unskilled user, and day to day maintenance will almost certainly be carried out by such users. There is a similar disparity within the end user population. While the users of a business computing system tend to have at least some computer-related experience, have completed at least some formal education, and are capable of making abstractions. Conversely, in a home environment, none of these factors may be assumed. This presents an interesting human-computer interaction challenge.

Much of the current research effort in home automation is directed toward finding appropriate network technologies. In many systems, specialized network technologies are used for certain purposes, such as mobile use or multimedia. In this case, the problem of integrating disparate network technologies must be addressed.

As has been noted previously, the users of a home automation network cannot be assumed to have any computing background or training. Therefore, many home automation projects aim to provide a network with the ability to configure itself automatically. In particular, when a device is (physically) added to the network, it must be automatically detected, and the system’s model of the network updated accordingly. This involves both registering the device’s presence, and determining the functionality that the device provides. The latter is generally achieved in one of two ways. The majority of systems export a description of the devices interface, which provides sufficient information for other devices on the network to contact the

device and cause it to perform operations. This is the approach taken by, among others, the Universal Plug and Play standard.¹ Another approach is for the device to make fragments of executable code available. Other devices can then download and execute this code in order to access functionality. This is the approach taken by Sun's JINI architecture;² unsurprisingly, the code transmitted takes the form of Java objects.

End-user programming

Office environments are the most common target for end-user programming systems. As with any design task, it is essential to define the population for which the design is intended. We have already described end users as users who do not necessarily possess any programming knowledge or experience; (Nardi 1993) offers an important addition to this definition:

End users are not "casual," "novice," or "naive" users; they are people such as chemists, librarians, teachers, architects, and accountants, who have computational needs and want to make serious use of computers, but who are not interested in becoming professional programmers.

This makes two important points. Firstly, while end users do not possess computing knowledge, they may well possess a high degree of knowledge in their particular domain, be it history, economics or medicine. Secondly, these users do not *want* to acquire computing knowledge in and of itself - they are busy being historians, economists and doctors. Hence, the problem of end-user programming is that of making the power and flexibility of programming accessible with as little effort as possible on the part of the user.

Many attempts to solve this problem take the form of improved interaction techniques, which purport to render the question of programming irrelevant. One common example is natural language systems; why, it is argued, would people want to program, when they can simply state their intentions in natural language? In this scenario, the computer would act as an assistant. Nardi argue that this vision is misleading, chiefly because the *programmer* is still

¹ Universal Plug and Play device architecture: http://www.upnp.org/download/UPnPDA10_20000613.htm

² Jini technology architectural overview: <http://www.sun.com/jini/whitepapers/architecture.html>

providing the assistant's functionality. She argues that advances in interaction techniques do not obviate the need for programming, and proposes that end user programming solutions should focus on task-specific languages and environments, and take into account working practice and other sociological factors, in addition to exploiting advanced interaction techniques.

Programming by demonstration

A common theme of both Programming By Demonstration and Visual Programming systems is to give a concrete representation of each concept in the programming environment. As Finzer and Gould (1993) put it:

Everything in the Rehearsal World is visible; there are no abstractions and only things that can be seen can be manipulated. Almost all of the designer's interactions with the Rehearsal World are through selection (with the mouse) of some performer [Smalltalk object] or of some cue [Smalltalk message] to a performer.

This is not to say that all state is visible to the eventual user of the script. The system discussed above includes an area known as the "wings"; objects placed in this area are visible to the designers, but not to the end users of the script.

Programming by example systems are distinguished from the simple macro systems present in many applications by the capability to *generalise*. A simple macro facility will allow the user a simple mechanism to repeat a sequence of actions exactly, without taking into account variations in context. More advanced macro facilities, such as those found in Microsoft Office (1997), allow the user to edit a generated macro to make it applicable in a range of contexts, but this still requires them to manipulate a traditional, textual programming language (in this case, Visual Basic).

In contrast, programming by example systems may generalise programs. One approach is to infer the required generalisation using artificial intelligence techniques. This method is attractive as it requires no effort on the part of the user, but is difficult to achieve in practice, and is best suited for systems dealing with limited problem domains. In particular, it is vital to include some way of rejecting incorrect inferences. Maulsby and Witten (1993) describe a

system, Metamouse, which uses AI techniques combined with user intervention. The tradeoffs associated with different degrees of intelligence in programming by example systems are addressed by Myers and McDaniel (2000).

```
Title:
/Nfs/buntingford/usr19/rgh22/work/docs/phd/toontalk.eps
Creator:
GIMP PostScript file plugin V 1.06 by Peter Kirchgessner
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.
```

Figure 1. ToonTalk (from Kahn 2000)

Another approach is to allow the user to indicate the way in which a program may be generalised. ToonTalk (Kahn 2000) is based on a “concurrent constraint programming language”, broadly similar to the logic programming language Prolog. The system is designed to appeal to school-age children, and to this end it is presented as a cartoon-like video game, with various components of the interface represented as animated characters (see Figure 1). Objects such as numbers and characters may be manipulated directly. To program, the user places a “robot” into the workspace, and tells it to watch while the user performs actions some input via direct manipulation. Once this training is complete, the robot will perform the specified actions whenever it sees objects identical to the original input. The user may watch the robot carry out the individual steps, to check that it is performing as expected, or instruct the robot to “fast forward” and only show the end result.

The system as described so far is equivalent to the simple macro recorders mentioned above. Its power is due to the simple yet expressive method of generalisation. It is possible to modify a robot to act on a wider class of inputs by removing concrete values from the “pattern” of recognised inputs, creating “wildcards” that may take any value. Facilities are also provided for conditionals, abstraction, encapsulation and communication, all presented in the

same cartoon-like way. Together, they allow children as young as five to create simple programs, while older children can create moderately complicated programs such as games.

ToonTalk programs, unsurprisingly, tend to be highly graphical, and make use of animation - the tool was designed to make the creation of such programs as easy as possible. However, for problems that do not have a strong graphical element, or require complex data structures, ToonTalk can become unwieldy. The pervasive use of metaphor, while arguably well suited for a pedagogical tool, would need to be examined carefully before being transferred to another application domain.

Visual programming

A *visual language* is described in Marriot *et al.* (1998) as “a set of diagrams which are valid *sentences* in that language, where a diagram is a collection of *symbols* in a two or three dimensional space”. This is reminiscent of Chomsky’s definition of (spoken and textual) language as a set of valid *strings* of symbols. Hence, the key distinction drawn between visual and textual languages is that textual languages are linear (sequential, one-dimensional), whereas their visual counterparts are expressed in a two- or three-dimensional medium. Visual Programming (VP) systems use visual as opposed to textual languages to express programs.

Blackwell (1996) surveys and categorises the perceived benefits (termed *metacognitive theories*) of VP, and notes a “remarkable consistency of metacognitive concerns amongst VP researchers”. The theories identified are based in many areas, including introspection by researchers, cognitive theory and folk psychology. While VP is not, as is sometimes claimed, a panacea that is better in all respects to conventional programming methods, there is nevertheless convincing evidence that VP is beneficial in certain specific ways.

VP systems are often based around direct manipulation, a technique more commonly associated with activities not traditionally thought of as programming. The main characteristics of direct manipulation are (to paraphrase Schneiderman 1983) continuous representation of the objects of interest, “physical” actions, rapid incremental reversible operations whose impact is immediately visible, and a layered or spiral approach to learning that permits usage with minimal knowledge. Consequently, many of the lessons learnt in the

design of direct-manipulation based Graphical User Interfaces (GUIs) may be profitably applied to VP.

A common technique employed in GUI design is metaphor; the system is made to mimic another, real-world, system. The advantage of a metaphor is to allow users to “learn by analogy”. However, an inappropriate metaphor, in addition to hampering learning by analogy, may cause users to make incorrect assumptions about the system. Furthermore, in some cases, a visual analogy may not provide significant benefits to comprehension (Blackwell and Green 1999).

In general, the application areas in which VP has been used most successfully are those where the problem in question has a large visual/spatial component, a notable example being GUI design. Integrated Development Environments (IDEs) are now used almost universally for this task. These systems combine a textual programming environment with visual tools to define the layouts of GUI components. While early systems limited the user to specifying simple, absolute layouts, modern systems are considerably more flexible, allowing users to specify complex relationships between layout parameters in a visual manner.

End-user programming in the networked home

While many research projects deal with *home automation*, few have considered end-user programming. As an example, the automated home environment created by Microsoft’s Easy Living research group includes a rule engine to allow the environment to act on sensor information; however, defining new rules involves writing or modifying a C program (Brummit 1999). A common view is that ubiquitous computing will obviate the need for programming; as mentioned earlier, this is at best a point of contention.

As mentioned above, it is imperative to form a clear picture of the intended users of any end-user programming system. In the case of an end-user programming system for an automated home, the users are the occupants of the home. This is a very wide group, encompassing a wide range of ages, occupations and levels of education. At first sight, it would appear that there are few, if any, useful generalisations that may be made about such a diverse population. However, closer inspections reveals important commonalities. In

particular, a user wishing to automate the control of a device *may be assumed to understand how to control that device directly*. This means that end-user programming languages for the home may profitably be built upon the basis of direct manipulation, such as the ToonTalk system mentioned above.

Graspable user interfaces

There is a growing body of research into designing physical objects to allow improved human-computer interaction. Such physical objects, or props, have numerous advantages over virtual objects. They are familiar to users, and provide rich affordances in a way that the user is already comfortable with. Physical objects are also subject to physical constraints. This is advantageous, in terms of communicating valid actions to the user, but limits the flexibility of props compared to virtual constructs. For example, the contents of a real folder is limited by size, and placing one folder inside another is problematic at best. These problems are not an issue with virtual folders in desktop computing systems. Similarly, while virtual objects may be modified instantly, in any way imaginable, physical objects are limited in this respect by their mechanical capabilities.

The Tangible Media group at MIT have produced a number of demonstration systems exploring the possibilities using physical and virtual objects in conjunction, within everyday architectural spaces (Ishii and Ullmer 1997). The group is interested in both props for user manipulation, and the exploitation of “ambient media” such as airflow and background noise as a continuous data channel that users may monitor passively.

Rekimoto *et al.* (2001) describe one of these systems, in which physical tokens (DataTiles) are combined with a display for virtual objects in an effort to offer the advantages of physical objects while retaining some of the flexibility of virtual ones. Transparent tiles are placed within a tray incorporating a display and sensors to determine which tiles are where. The virtual portion of the tile may then be shown behind the physical tile, on the appropriate section of the display. This allows the tiles to be used as “graspable windows for digital information”

This technique may be used to combine high-resolution (printed) but static data, such as a map, with low-resolution dynamic data, such as the current location of objects. It may also be used to afford certain physical actions. The paper gives the example of a rotational control, that combines a circular groove acting as a guide for a stylus with an appropriate graphical component to allow an angle, which may be mapped to a value, to be entered. Simple aggregate effects may be achieved by placing tiles next to one another, but this area is as yet under-explored and offers much scope for further work.

AutoHAN infrastructure

The AutoHAN project (Saif *et al.* 2001) is an ongoing project in the Computer Laboratory, working on technologies to enable a pervasive, self-configuring network in a home context. It provides a solid foundation to support a home network architecture, including support for user interaction devices. IP is used as the basis of the network architecture, allowing heterogeneous link-layer technologies to be employed. The technologies currently include Warren (Greaves 1997), a variety of ATM designed to allow simple end-point devices, and Ethernet. TCP and UDP are used as transport protocols.

Title:
(AutoHAN.eps)
Creator:
Adobe Illustrator(R) 8.0
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Figure 1.The AutoHAN architecture

At the application level, communication is performed over several higher-level protocols. Chief among these is HTTP. At present, HTTP/1.0 is used in most applications, as this provides sufficient flexibility without the complexity of HTTP/1.1, which requires the

implementation of a large set of relatively complex features. HTTP/1.0 is a request/response protocol using the message format described in IETF RFC 822. This allows request and responses to have a *type* or *method*, describing the type of request or response, a set of arbitrary *name/value pairs*, and a *body* containing arbitrary data. Conventionally, HTTP methods `<SMALL>GET</SMALL>` and `<SMALL>POST</SMALL>` are used to retrieve documents from and send queries to a web server. These methods are used in AutoHAN to query and update DHAN, an XML based registry.

SOAP is a Remote Procedure Call protocol based around HTTP, using an extended set of methods. Objects that provide a SOAP interface necessarily run a server to listen for, and reply to, requests. GENA is a complementary HTTP-based protocol for eventing. Applications run a server, and then subscribe to certain classes of event by sending an appropriate request, containing the URL of their server to a Subscription Arbiter (SA). The SA then sends a request to the application whenever an appropriate event occurs. Both SOAP and GENA are used extensively in the Universal Plug and Play architecture, and are described in that specification.

XML is widely used as a data presentation format in AutoHAN.³ Devices are described in XML, as are the payloads of SOAP and GENA messages. XML was chosen as it is flexible, portable and human readable, and, most importantly it may be extended to accommodate future changes in the data's structure.

To enable further research to utilise this infrastructure, the appropriate high-level protocols (notably GENA and HTTP) have been implemented in several languages. As may be inferred from the above descriptions, this involves supporting the implementation of both clients and servers. C++ and Java interfaces have been created and used extensively. The object-oriented scripting language Python (see van Rossum) may also be employed, as it has an extensive standard library that includes HTTP servers and clients and an XML parser. The language has proved to be useful for rapid development, but it lacks the runtime efficiency for some applications.

³ Extensible Markup Language (XML) 1.0 (second edition): <http://www.w3.org/TR/REC-xml>

Other members of the AutoHAN group are working on developing a general-purpose language, Iota, which has many features useful in developing AutoHAN software. It is a statically-typed impure functional language, similar to ML (Milner *et al.* 1997), with sophisticated fine-grain concurrency primitives based on the *Pi* calculus (Milner 1999). A major feature of the language is that XML data is a primitive datatype, with rich syntactic support, particularly pattern matching. In addition, Iota is intended to be used with a standard library tailored to a specific application domain. The initial version of the language, Iota.HAN, includes a library to support the development of AutoHAN infrastructure and applications, providing capabilities such as HTTP. Iota.HAN is primarily intended for use by device manufacturers in order to create AutoHAN applications to ship with consumer devices. However, it is sufficiently general to be useful in implementing AutoHAN infrastructure components. The design process of the Iota language is discussed in (Blackwell and Hague 2001).

Media cubes

The part of the project being discussed here deals with designing a novel user interface that will enable end users to control networked devices, both directly and indirectly (automation/programming). The current approach centres on novel interaction devices dubbed Media Cubes, as shown in Figure 3.

```
Title:
/Nfs/buntingford/usr19/rgh22/work/docs/phd/oldcubes.eps
Creator:
GIMP PostScript file plugin V 1.06 by Peter Kirchgessner
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.
```

Figure 1. Prototype media cubes

The Media Cubes provide a concrete representation of abstract concepts; for example, a cube could represent the concept of reaction (“Do X when event Y happens”). This representational capacity also helps the user manipulate devices that are not physically present, such as devices in other rooms. Similarly, in the networked home, some devices may have no “front panel” user interface, or may be implemented as software objects. The Media Cubes are designed to provide an suitable tool for users to work with such devices. They may also be used as a physical representation of media streams or content, as in (Ullmer *et al.* 1998) and (Lamming *et al.* 2000).

Media Cubes are introduced to users as simplified remote controls, in the sense that, while familiar remote controls offer many functions for a single device, a Media Cube offers a small number of functions, but may be used to control any device. Once the user is comfortable with using the cubes in this way, the concept of placing cubes together and using them in combination is introduced, allowing the user to abstract functions over time and space. This provides a means for the user to progress smoothly from direct manipulation to programming.

Figure 3 shows the current prototypes of the media cubes, designed by Alan Blackwell and Daniel Gordon. Each cube has the following external features:

- A single tri-colour LED, mounted on the topmost face, allowing limited feedback regarding the state of the Media Cube.
- A single button, also mounted on the topmost face.
- An IR transmitter and receiver, providing two-way communication with a networked base station.
- Four coils, mounted on the vertical faces of the Media Cube. The coils are used to detect the proximity of other Media Cubes (see below).

Each cube contains a small microprocessor. This pulses each coil around sixteen times a second, and monitors the current through each coil when it is not being pulsed. If another cube is adjacent to the coil being pulsed, a current is induced in the corresponding coil of the second cube, and detected by its microprocessor. In this way, each cube can calculate the set of faces that are adjacent to other cubes. This information is relayed to an infrared base station, where it

is translated into a GENA event, and hence may be acted upon by software components in the AutoHAN architecture

Each cube has a hard-coded, unique ID; the cubes are identical in all other respects. A software layer maps meanings onto individual cubes. The cubes are labeled to indicate this meaning to the user. In some cases, the meanings of certain faces or cubes may change as they are used - in this case, re-writable labels are provided.

A piece of software known as the Cubes Proxy keeps track of the status of the cubes, and holds the representation the constructs created, via the cubes, in the Media Cubes language (see below). Like many AutoHAN software components, the Cubes Proxy communicates with the network at large solely via GENA events, meaning that it may be run on any machine with an IP connection to the network.

The Cubes Proxy subscribes to events from one or more base stations, and hence receives events corresponding to each of the transmissions from the cubes via HTTP. These events are then decoded to determine which cubes are interacting, and which of their faces are adjacent. Each individual cube is associated with a proxy, i.e. a software representation, via its ID. These representations are used to determine the semantic meaning of the particular interaction at hand, which may:

- ❑ Change the state of the Cubes Proxy, for example by defining a new script or changing the value of a cube.
- ❑ Cause one or more GENA events to be broadcast, allowing the cubes to control other software components and devices.⁴

Currently, the Cubes Proxy is written in C++, and runs on a Linux PC. Each physical cube recognised by the system has a corresponding C++ object, associated by the unique ID of the cube. The object is an instance of some class that implemented the “CubeProxy” interface, which defines methods to handle events where a button is pressed, and where the cube has been placed next to another. When an event is decoded, it is passed to the appropriate proxy object, along with the proxy object for the other cube in the latter case. As any class that

⁴ Of course, a single interaction may do both.

implements the appropriate interface may be used as a proxy for a physical cube, additional behaviours may be readily added to the system.

When used in combination, the **Media Cubes** constitute a medium in which a program may be expressed. Our aim is to construct a language that may be effectively used by end users to express home automation programs, or scripts, in that medium.

Currently, a minimal demonstration language has been implemented. This language is built on three types of cube; an *event cube*, which has a GENA event associated with each face, a *timer cube*, which has a delay (10 seconds, one minute, etc...) associated with each face, and a *contingency cube*, which has two active faces, labelled “Do...” and “When...”.

The event cube may be used as a direct manipulation device. The faces are viewed as an ordered sequence, and pressing the button on the cube causes the next event in the sequence to occur. For example, if the faces of the cube are associated with the events “Connect CD Player”, “Connect Radio Tuner”, “Connect DVD Audio”, and “Connect TV Audio”, pressing the button repeatedly would allow the user to cycle through the various audio sources. When used in combination with the other two cubes, very simple abstractions may be constructed. First, an event from the event cube is associated with the “Do...” face of the contingency cube by placing the appropriate faces together and pressing the buttons on the cubes. A delay is similarly associated with the “When...” face. Then, after the specified delay, the specified event occurs.

Future work

The existing demo system may be profitably improved with a small amount of effort. Currently, contingencies only function correctly with delay events. A minor modification would allow events to be contingent on any other event, allowing programs such as “mute the audio when the phone rings” to be constructed. Also, the current system has only rudimentary type checking, which allows incorrect programs to be constructed. Again, minor modifications to the system would allow the integration of a more advanced type system. With these additions, the system will be more than sufficient to support the research into **Media Cubes** languages described below.

While working with the AutoHAN system, it has become apparent that the use of a full system, including hardware, is problematic, and much time is spent on maintaining this system at the expense of more productive work. To alleviate these problems, a number of software simulations of real devices will be developed. These will communicate via the standard GENA architecture, and hence a system developed with the virtual devices should communicate with physical devices with little or no modification. Virtual devices also have the advantage that they may be created and modified as required, allowing a variety of network configurations to be used. Simulations may also be created for devices that have not yet been implemented, allowing a far broader range of tasks to be addressed by users during testing.

A MEDIA CUBES LANGUAGE

Using the work mentioned previously as a basis, a language based around the Media Cubes will be designed. We are currently considering two alternative programming paradigms for this language (Blackwell and Hague 2001); the cubes can either be taken to be analogous to words in a spoken or written language (linguistic abstraction), or they can be taken to be representations of elements in the users world model (ontological abstraction).

The ontological paradigm associates cubes with “natural categories” in the user’s mental model of the system, and the faces of those cubes represent interpretations or interactions of those categories. For example, an Event cube represents a change in system state, such as a doorbell ringing, and supports interpretations such as “on” and “off”. Similarly, a Channel cube represents a media stream and has interpretations to support routing of the channel as desired. The intention is that, by providing users with a set of tokens that represent familiar or “sensible” ontological categories, the language provides support for reasoning about the abstract concepts involved.

The linguistic paradigm is similar to object-oriented textual and visual languages. Cubes have associated data, and have faces that allow access to and modification of this data. An example would be a cloning cube; this cube would have a face labelled “clone” that, when placed next to some face of another cube, stores the value of that face. Another face of the cube may then be used as if it were the cloned face of the second cube. An extension of this

cube would be a collection cube, which would allow a single cube face to take on the properties of multiple faces drawn from other cubes. For this to be practical, a type system may be necessary in order to ensure that cube faces are not combined in a nonsensical way.

An important feature common to both paradigms is the ability to associate a cube with a physical device, simply by placing a face of the cube adjacent to a sensor on the device. This allows physical objects to be referred to directly, reducing the level of abstraction necessary to construct programs.

Two initial language designs will be produced, one for each of these paradigms. These will then be implemented within the framework already created. The prototypes will then be tested on users, and the results of these tests will be used to produce a second version of the language. This revised language will be based on one of the possible paradigms, but is likely to include ideas from both initial prototypes. An orthogonal question that will also be addressed during the user testing stage is the degree of computational flexibility required by the language; specifically, is it necessary for the language to be Turing powerful? While current domestic device interfaces are not, it may be the case that it is beneficial for a more general interface to be. Conversely, a computationally complete language may prove confusing or otherwise inappropriate for this application area. A related issue is that of the provision of higher-order functions, which can fit into either paradigm.

MULTIPLE END-USER PROGRAMMING LANGUAGES

In addition to the tactile language developed with the Media Cubes, other approaches to end-user programming in the home and other ubiquitous computing situations will be investigated. While the Media Cubes provide a rich input medium, their output capabilities are severely limited. In addition, the number of physical cubes available limits the vocabulary of the Media Cubes language. To overcome these limitations, a visual language for use with conventional displays such as televisions will be developed. One use of this language will be to view programs created via the Media Cubes, while another use may be to modify existing programs. The latter use will require some form of input. While a mouse may be used, pen or touchscreen input would seem to be more appropriate given the application area - this suggests a device

similar to the WebPads that are currently coming to market. A keyboard is unlikely to be readily available, so textual input should be made optional or avoided altogether. It is important to note that, while a visual programming language overcomes the stated limitations of the Media Cubes language, it does not possess some of the advantages of that language, such as the smooth transition from direct manipulation. The two languages are complementary.

Other types of media may be explored. Video-based systems, such as the In/Out board and BrightBoard described in Stafford-Fraser (1997) may be appropriate for certain areas of the home. These systems offer an unobtrusive input medium with a minimal hardware infrastructure, but must be carefully designed to ensure robustness. Locations-based systems such as those developed by the AT&T Sentient Computing project offer another possibility for a pervasive medium for programming in a home network. It is possible that several of the media may be fruitfully combined.

While Iota is being developed to enable software engineers to easily create AutoHAN applications, it may be profitable to consider an alternative textual language designed for a less skilled group of users. This language would be used to “plumb together” AutoHAN components in a home network. It is expected that this task would be performed either by a confident end user, or a technician brought in to install a device, much as today an electrician may be called in to install a wall socket by someone who is not confident doing the job themselves. In either case, it is not necessarily true that the end-user will have the degree of computer science knowledge required to fully understand a concurrent, statically typed functional language. A more accessible language would fill a similar niche to scripting languages such as Perl (Wall *et al.* 1996) in conventional system administration, in that ad-hoc solutions to problems may be produced quickly and easily. Like Perl, the language will not be designed for the implementation of large-scale or general applications, and is unlikely to be the best choice for such tasks.

A FRAMEWORK FOR MULTI-LANGUAGE SYSTEMS

The previous section details a number of languages designed to allow end-users to program their home network. These languages each have different, but complementary, features. Hence, substantial gains may be made by allowing the languages to interoperate. A language-neutral representation will be developed, and used as the basis of the language implementations. Unlike portable executable formats such as Java bytecode, this representation should allow high-level source code to be generated. This will allow a program to be created in one language, and modified in another - a valuable capability when the preferred method of creating programs, e.g. the Media Cubes, does not provide a convenient interface for program modification. Ideally, systems languages such as Iota would be integrated into the framework, to allow seamless integration between provided applications and end-user programs. A significant problem to be addressed is that of translation between disparate languages with different paradigms a problem that has proved difficult in the past. For example, Microsoft's .NET architecture allows tight integration between a variety of languages, but requires all languages to fit into an object-oriented structure, at least where they interact with code in other languages.

XML may be used for the language representation. This would allow individual language implementations to leverage the large body of tools, libraries and knowledge regarding XML processing. In addition, presentation of the program may use technologies such as Cascading Style Sheets,⁵ or XSLT,⁶ to present the program in a conventional web browser. The latter technology may be used to transform one XML document into another in an arbitrary way. It would be straightforward to transform a program into SVG,⁷ and XML based vector graphics format, allowing for easy and flexible graphical presentation.

⁵ Cascading style sheets level 2 specification: <http://www.w3.org/TR/CSS2/> .

⁶ XSL Transformations (XSLT) version 1.0: <http://www.w3.org/TR/xslt>

⁷ Scalable vector graphics (SVG) 1.0 specification: <http://www.w3.org/TR/SVG>

An execution engine for the language-neutral format would allow programs developed in any supported language to be executed. This engine would handle issues such as event subscription and reception, and handle received events according to the appropriate stored program. The programs would be submitted over the network by editors or proxies that implement specific language interfaces. Programs may also be retrieved for display or modification. HTTP offers an appropriate interface for submitting new programs and retrieving stored programs, but may not be suitable for modification of stored programs. The engine may be integrated with a general AutoHAN policy enforcement mechanism, allowing scripts created by different users, or in different circumstances, to have an appropriate level of access to the system. This would prevent, for example, a child creating a script that circumvented a spending restriction for pay-per-view TV, or anyone from creating a script that muted the burglar alarm.

Pervasive networking of domestic appliances provides a wealth of possibilities. Developers will anticipate some of these possibilities, but many will be novel and unexpected. Hence, the provision of end-user programming adds significant utility to a networked home. The AutoHAN project aims to provide a consistent framework in which everyone may be able to set up and use a home network in a convenient and accessible way.

Acknowledgements

The AutoHAN project is led by David Greaves. Rob Hague is sponsored by AT&T Laboratories – Cambridge Ltd. Alan Blackwell’s research is funded by the Engineering and Physical Sciences Research Council under EPSRC grant GR/M16924 “New paradigms for visual interaction”.

References

- Blackwell, A. F. and Green T.R.G. (1999) “Does metaphor increase visual language usability”, *IEEE Symposium on Visual Languages*, pp. 246-253.
- Blackwell, A.F. and Hague, R. (2001) “Designing a programming language for home automation”, *Proceedings of the 13th Annual Workshop of the Psychology of Programming Interest Group*, pp. 85-103.
- Blackwell, A.F. (1996) “Metacognitive theories of visual programming: what do we think we are doing?”, *Proceedings IEEE Symposium on Visual Languages*, pp. 240-246.
- Brummit, B. (1999) “Easy living”, *seminar given at Microsoft Research*, Cambridge.
- Cypher, A. (ed.) (1993) *Watch What I Do: Programming By Demonstration*, MIT Press.
- Engelbart, D.C and English, W.K. (1968) *A research center for augmenting human intellect*, Thompson Books.
- Finzer, W.F. and Gould, L. (1993) “Rehearsal world”, *Watch What I Do: Programming By Demonstration*, MIT Press.
- Greaves, D. (1997) “ATM in the home area network” paper presented at the *IEE Colloquium on ATM in Professional and Consumer Applications*.
- Ishii, H. and Ullmer, B. (1997) “Tangible bits: towards seamless interfaces between people, bits and atoms”, *Proceedings of CHI '97*.
- Kahn, K. (2000) “Generalizing by removing detail: how any program can be created by working with examples”, *Your Wish is My Command: Giving Users the Power to Instruct their Software*, Morgan Kaufmann.
- Lamming, M., Eldridge, M., Flynn, M., Jones, C. and Pendlebury, D. (2000) “Satchel: providing access to any document, any time, anywhere”, *ACM Transactions on Computer-Human Interaction*, vol. 7, pp. 322-352.
- Marriot, K., Meyer, B. and Kent B. (1998) “Wittenburg: a survey of visual language specification and recognition” *Visual Language Theory*, vol. 2, pp. 5-85.
- Maulsby, D. and Witten, I.H. (1993) “Metamouse: an instructable agent for programming by demonstration”, *Watch What I Do: Programming By Demonstration*, MIT Press.
- Microsoft Press (1997) *Microsoft Office 97 Visual Basic Programmer’s Guide*.
- Milner, R. (1999) *Communicating and mobile systems: the π -calculus*, Cambridge University Press.
- Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997) *The Definition of Standard ML (revised)*, MIT Press.
- Myers, B.A. and McDaniel, R. (2000) “Demonstrational interfaces: sometimes you need a little intelligence; sometimes you need a lot”, *Your Wish is My Command: Giving Users the Power to Instruct their Software*, Morgan Kaufmann.
- Nardi, B.A. (1993) *A Small Matter of Programming*, MIT Press.
- Rekimoto, J., Ullmer, B. and Oba, H. (2001) “Datatiles: a modular platform for mixed physical and graphical interactions”, *CHI 2001 Conference Proceedings*, pp. 269-27.
- Saif, U., Gordon, D. and Greaves, D (2001) “Internet access to a home network”, *IEEE Internet Computing (February)*, pp. 54-63.

- Schneiderman., B. (1983) "Direct manipulation: a step beyond programming languages", *IEEE Computer (August)*, pp. 57-69.
- Stafford-Fraser, J.Q. (1997) *Video-Augmented Environments*, Ph.D. Thesis, University of Cambridge.
- Sutherland, I.E. (1963). "Sketchpad: a man-machine graphical communication system", *SJCC*.
- Ullmer, B., Ishii, H. and Glas, D. (1998) "MediaBlocks: physical containers, transports, and controls for online media", *Computer Graphics Proceedings (SIGGRAPH '98)*.
- van Rossum, G. - Computer programming for everybody: <http://www.python.org/doc/essays/cp4e.html>
- Wall, L., Christiansen, T. and Schwartz, R.L. (1991) *Programming Perl*, O'Reilly and Associates, Inc.
- Weiser, M. (1991) "The computer for the 21st century", *Scientific American (September)*, pp. 94-110.