In Proceedings of the 2001 IEEE Symposia on Human-Centric Computing Languages and Environments (© IEEE), pp. 150-157.

AutoHAN: An Architecture for Programming the Home

Alan F. Blackwell and Rob Hague
Computer Laboratory
University of Cambridge
{Alan.Blackwell, Rob.Hague}@cl.cam.ac.uk

Abstract

AutoHAN is a networking and software architecture that enables user-programmable specification of the interaction between appliances in a domestic house. This concept represents an immense challenge for End-User Programming. The characteristics of the potential user population are far broader than any other population of end-user programmers. It is therefore essential to approach the design of the programming environment from a well-founded perspective of cognitive ergonomics and user modelling. We have created a novel programming language, Media Cubes, which is aimed at the same user population as existing domestic remote controls. Moreover, we have applied a cognitive model of programming behaviour in order to specify a language that gives the advantages of direct manipulation as well as the power of more conventional languages.

1. Introduction

The private home is likely to become the most challenging frontier for end user programming techniques over the next 10 years or so. We justify this claim on the basis of three trends: changes in our homes, changes in the demands of programming, and resulting changes in the population of end user programmers. First, our homes are changing rapidly with the introduction of microprocessor controls into many domestic appliances. Microprocessors are increasingly replacing electromechanical technologies even in simple appliances (toasters, boiler controls), as well as enabling new generations of complex digital home technologies (CD players, satellite TV, digital answering machines). The economics of appliance manufacture suggest that both these trends will only continue.

Secondly, new programming demands are already being introduced in the home as a result of domestic digital automation. Successful office software products are notoriously plagued by "feature-creep". This arises because software distribution costs are independent of the number of features, so manufacturers continually add features to maintain user interest. As the cost of memory in embedded microprocessors falls, feature creep also effects domestic appliances – it costs nothing to ship a product with additional software features, so manufacturers add them. But additional actuators and sensors are expensive, so the features are most likely to be abstract functions – personalisation, macro commands, and other *programming* functions.

We argue below that this type of function is not only called "programming" in casual terminology, but will come to resemble more conventional types of end-user programming, especially in its cognitive consequences and resulting usability challenges. This leads to the third trend in the challenging frontier of end user programming for the private home – the domestic context addresses the widest possible range of previous programming experience and cognitive skill. Most previous end user programming research has focused on the sort of programming tasks that might be attempted by office workers or other information professionals. A few research projects address the demands of programming in educational contexts, where a wider range of prior cognitive skills might be encountered. But the domestic context encompasses the whole population of Western countries, including people who may never have worked in an office or completed their schooling. We contend that all normal adults, whatever their level of schooling or computer experience, are competent to use a remote control. We have therefore set ourselves the challenging task of producing a programming language that is accessible to users whose skill with abstract tasks may not extend any further than the operation of a remote control.

1.1. Home Networking

The programming demands introduced by the abstract functions in any single domestic appliance do not yet (fortunately) require a full featured programming language. Programming a VCR, although relatively simple in algorithmic terms, is proverbially considered to be the most challenging programming task in the home (many computer scientists would not consider this to be true "programming", despite the popular use of the term – we address this below). Nevertheless, some new categories of domestic appliance are rapidly developing more recognizable features of general purpose programming functionality – the ability to record macro-like sequences of front panel operations, for example, or the creation of directories and indexed data structures on new digital audio media.

But the potential complexity of domestic programming is massively increased where digital devices interact with each other – especially if the devices are manufactured by different companies, so that it is the user who has to specify their interaction. This is not yet a highly significant factor in domestic electronics. But home networking technologies are rapidly being deployed, if not in the average person's house, then at least in many research environments. Large companies are experimenting with home networking. Nascent standardisation bodies are competing to define networking and communications protocols (e.g. [1,2]) by which appliances can communicate with control devices and with each other.

As noted, "programming" a VCR may not be not real programming in the eyes of most computer scientists. But if you wanted to instruct your VCR to start recording from the front door security camera for a period of 5 minutes after the time that your security system records someone pressing the front doorbell, this seems a lot more like real programming. So the significance of home networking technology is that programming in your house may suddenly become a great deal more complicated. If your VCR can talk to your home security system over the home network, how will it know what to say? Standardisation efforts are partly focused on trying to define "sensible" standard functionality for all possible combinations of appliances. But we believe such an approach is doomed to failure. The combinatorial explosion of different device interactions, and the notorious difficulty of defining software standards for programming at the level of domain semantics (compare the "business object model" of the 1990s) mean that users of networked domestic appliances are very likely to find themselves defining functionality through some level of home programming.

1.2. The AutoHAN project

These issues are the main concern of the AutoHAN project. A major aim of AutoHAN is to provide programming facilities that will be accessible to a domestic end-user whose previous experience of specifying digital control functions may be limited to the operation of a remote control. Our aim has been not simply to speculate about the theoretical characteristics of such a system, or even to specify the way it would operate and the programming language it would use, but actually to build an operational home network with an integral programming language that can be evaluated experimentally with a range of typical users.

Our experimental network includes a range of next-generation appliances such as CD players, digital telephones, video cameras etc. All of these appliances have been modified with network interfaces so that they can communicate with each other, and with central server facilities. Communication protocols have been developed to support abstract interaction at a level suitable for general purpose programming. Finally, a new kind of programmable remote control has been developed as the basis for a novel end user programming language. We describe these facilities in the rest of the paper.

2. Cognitive demands of domestic end user programming

We believe that all research in end user programming should be conducted with the priorities of user-centred design. If a research prototype purports to be addressing the needs of a specific user population, the researchers should analyse the needs of the user first, and develop their technical solution second. This is especially true of the domestic context, where users expect that the products they buy have been designed with ergonomic and usability criteria in mind. Users of commercial programming tools may not have such high expectations, as they are more accustomed to arbitrary technical solutions assembled with only a passing awareness of cognitive ergonomics.

Despite the obvious importance of focusing on user requirements, much of the research literature related to home automation and networking only mentions this factor in passing. The IBM pervasive computing group, in their recent textbook on pervasive computing [3], only mention usability in a cursory fashion – they observe that pervasive computing devices should be made both "convenient" and "intuitive" [3, p.23], but do not offer any further guidance on how this is to be achieved. The Microsoft "Easy Living" research group have created a prototype living room with a range of networked appliances. If the occupant wants to define new behaviour, he or she can walk to the computer in the corner of the living room and write an appropriate program – in 1999, this was accomplished by writing in C [4].

2.1. Direct manipulation

The first author of this paper has for some years taken the approach that end user programming research must be informed by the research methods of cognitive psychology, in order to understand the user's needs, and then to design appropriate solutions based on these cognitive requirements. This has led to important observations regarding the usability strategies of the AutoHAN project. One of them is derived from the first author's discovery that direct manipulation is far more important than pictorial metaphors for assisting end-users to learn a novel programming language [5,6]. The AutoHAN system has been designed to place the greatest possible priority on direct manipulation – users program the system by directly manipulating physical blocks, not just representations of blocks on the computer screen.

2.2. Notational abstraction

A further cognitive principle in the design of AutoHAN is to recognize that all programming behaviour involves the user making an investment in abstraction. In terms of domestic appliances, abstractions are the aspects of the appliance that we can't see or touch. But when a computerised appliance contains a user interface, the user is expected to interact with abstractions – by means of what he or she sees and touches in the interface. To this end, the interface includes representations of abstraction, or signs. Users see and manipulate those signs in a systematic way. If they become sufficiently familiar with the conventions, they may even think of the sign as the object, forgetting that they are only manipulating signs (abstractions are not things, and can never be touched).

Two research issues arise from this. The first addresses the status of abstract behaviour: how can the user interface of an appliance (let's say a MiniDisc player) include things we can't touch? Briefly, this can occur either because the user is asking the appliance to do something in the future (like recording a radio programme – as it is in the future, we can't interact with it now), which we call an *abstraction over time*; or alternatively because the user is referring to a number of entities (like a playlist of music to be played at a party) which we call an *abstraction over a class* of entities.

The representations of these abstractions are notational systems, in the sense of Green's Cognitive Dimensions of Notations [7,8]. The user interface of the appliance includes a notation (a visible representation of the abstractions) and a physical means of manipulating the notation. Although very simple, the two types of abstract situation (over time or over a class) form the core of all programming languages, and also of all professional programming activity. It is on this basis that we claimed in the introduction that there is a relationship between "programming" a VCR and a "programming language".

2.3. Characterising abstraction users

The characteristics of end-user notational systems have more often been found in our offices than in our homes until now. Every office worker is expected to interpret, manipulate or create specialised abstract notations related to their work: timetables, flow diagrams, decision trees, work rosters and many more. If we use computers in our work, almost every type of software application includes notations that allow us to create and manipulate complex and powerful abstractions. The spreadsheet notation, for example, is almost as powerful as a general purpose programming language – which is probably the most challenging abstraction handling notation that any human ever has to deal with. But it's not just professional programmers or spreadsheet users that face challenges in computer abstraction.

Professional computer users (especially programmers) are very experienced at creating and manipulating abstractions. Is it too difficult for homeowners to attempt such a challenging intellectual activity? In a recent empirical study, we addressed this question directly [9]. We surveyed the office contents of secretarial workers who were not confident computer users, and compared them to the offices of computer science researchers. We found that the secretarial workers were always creating and maintaining abstractions: folders, drawers, cabinets, file boxes and binders. Computer scientists also use such abstractions – the two groups were perfectly comparable. But in the computer environment, computer scientists were likely to create at least one computer abstraction (directories, folders, macros and style) for every paper abstraction in their office. Office workers were incredibly reluctant to invest effort in computer abstractions. It's not that office workers are unable to deal with abstract notations – it's just that computers don't serve their needs as well as paper does.

2.4. A model of abstraction investment

We have modeled the cognitive aspects of this phenomenon by adapting Kahneman and Tversky's Prospect Theory – a model of the psychology of investment decisions [10]. The experimental subjects in our study of office workers were not investing money (as in Kahneman and Tversky's work), but they were investing their time and attention in creating abstractions. This is true whether the abstraction is a series of labels in a filing cabinet drawer, a word processor macro, a programmed recording time in a MiniDisc player, or a complex software application.

In each case the abstraction developer invests some attention in programming work that could otherwise be spent in achieving the job itself. Furthermore, the abstraction route is risky. It may not work, it may work partially (it contains a bug), or it may turn out to be inappropriate to what is actually needed when the time comes to execute it. This risk of failure is just like the risk of gambling losses in the situations investigated by Kahneman and Tversky. We have created a cognitive simulation of this phenomenon in which a simulated agent makes investment decisions about a simple abstraction – the search and replace dialog in a word processor [11].

2.5. Theoretically grounded end-user languages

This theoretical approach to end-user programming as the facilitation of abstraction investment has already been applied to a very different research project, in the application domain of office work. The SWYN system

described at a previous end-user programming workshop [12] provides additional abstraction management facilities within the functional area of scripting for word processing tasks [13].

Someone working with a word processor might regularly create abstractions over a class (e.g. defining a search and replace operation, or a paragraph style) or abstractions over time (e.g. a repeating keyboard macro). SWYN allows users to minimize their initial investment in abstraction, by defining the abstractions they need through direct manipulation of example strings (and hence programming by example [14]). It also allows the user to assess the degree of risk when the abstraction is executed, by giving direct feedback in response to user actions on the notation.

3. The Media Cubes language

The AutoHAN project has taken the same theoretical approach to supporting end user programming as in SWYN, but has applied it in the domestic context. The *Media Cubes* language allows the user to associate behaviour directly with a concrete representation. The user can also directly observe the effects of abstract actions that are initiated by manipulating the concrete representation.

This representation is a rather unusual programming language. It is not displayed on the screen of a computer, but is a fully tangible representation of the abstract situation. It is tactile, not in the sense of Repenning's "tactile programming" [15], but in the sense that the user can touch physical "iconic" representations, as in research by Ishii and Ullmer [19] for non-programming applications. The elements of the Media Cubes language are literally cubes – currently the prototypes are made out of wood. Two prototype Cubes are shown in figure 1. Each Cube contains a variety of sensors and transducers, a low-power PIC microprocessor, three AA-size batteries, and a motion sensing device allowing it automatically to become quiescent when not being manipulated. This delivers a battery life on the order of months.

Figure 1 – "Media Cube" prototypes

The user interface of the individual Cubes is as simple as possible. We describe them as "one-button remote controls", and encourage users to think of them in this way. Each Cube has a single button, a single LED which can be used to confirm status, and a piezo-electric transducer which can be used to notify the user of local state changes.

The Media Cubes are also equipped with several interfaces to other AutoHAN components. Each Cube includes an array of infrared receivers and transmitter, so that it can communicate with the AutoHAN network via IR ports in each room. In future iterations of the prototypes, we hope to replace this communication channel with a single-chip implementation of Bluetooth, and are working with a local Bluetooth developer to that end.

Each Cube also includes induction coils on four faces of the Cube. These act as short-range antennae, able to detect direct proximity of another Cube. All Cubes transmit periodic pulses through these antennae whenever they are not in the quiescent state (i.e. when the motion detector recognises that they are being held in the hand of a user). In the intervals between pulse transmissions, each Cube listen for pulses from other Cubes. When a pulse is received, the Cube notifies the network that this has happened, identifying the faces that have come into contact. The local state change is notified to the user by emitting a click from the piezo-electric transducer.

The induction coil antennae can also be used to establish a relationship between an appliance and a Cube. By holding one face of a Cube against the front of an appliance, the Cube can be *associated* with some function of that appliance – either temporarily, or for as long as the user keeps the Cube in that state (it can be associated with a different appliance by repeating the procedure with that appliance).

3.1. Direct manipulation of Media Cubes

By describing Media Cubes as "one-button remote controls", we intend that an individual Cube should be regarded by the user as an intuitively transparent direct manipulation interface to some appliance function. If a Cube represents the "play/pause function", for example, the user could associate that Cube with a CD player, and pressing the button would alternately place the CD player in play and pause mode. If the user then changed the association of the same Cube, by holding it against a VCR, it would thereafter control the play/pause function of the VCR.

This is a new paradigm for interaction with abstract functions of home appliances. The currently established paradigm is that the user has many remote controls in the house, each dedicated to a single appliance, but with one remote control providing access to many abstract functions and attributes of the appliance. In contrast to this paradigm, each Media Cube represents a single abstraction, but can implement that abstraction with respect to many different devices.

3.2. Abstract behaviour of Media Cubes

Although users may become familiar with the behaviour of individual Media Cubes by using them as one-button remote controls, the true power of the language is realised through composition of these basic abstractions. In terms of the psychological requirements of abstraction investment, the user is able to gain a better appreciation of the investment risk associated with individual abstractions by applying them in a direct manipulation situation before extending them to compound risk situations.

Cube abstractions are composed by placing two (or more) Cubes next to each other, and instructing AutoHAN to store this configuration of Cubes. These configurations literally act as programs. They are stored as scripts in the AutoHAN registry, and can be used to specify arbitrarily abstract behaviour – either executing home automation processes at some time in the future, or extending a process across a range of different situation classes.

As every Cube has a unique identifier, and each face of every Cube can also be identified, the combination of Cubes and neighbouring faces can be used to define the syntactic form of a reasonably powerful programming language. In fact, they share the rich syntactic potential of many visual programming languages, because the four possible adjacency relations between tokens allow a 2D arrangement of Cubes, rather than the 1D arrangement of textual programming languages having only two adjacency relations between tokens (before and after).

The rich syntactic potential of an arbitrary set of Cube tokens, each having up to four faces "inflecting" the basic token, allows many possible styles of programming. Our only constraint in choosing one has been to provide a cognitively plausible path for the user from the direct manipulation experience of using the Cubes as remote controls. This learning path should lead first to an appreciation of an individual Cube evolving from a representation of a specific appliance function (in the first instance) to a more general abstract representative of the appliance that can be used in a program. The second phase of the learning path is to utilise further Cubes that have completely abstract interpretations – this is described further in the later section on programming paradigms for Media Cubes.

4. AutoHAN infrastructure

The AutoHAN infrastructure has been developed to support both the direct manipulation and abstract programming functionality described above. This has been a considerable research effort in its own right. A more comprehensive technical description has been published elsewhere [16]. Here we present an overview of the infrastructure at a level sufficient to explain its properties as an end-user programming environment.

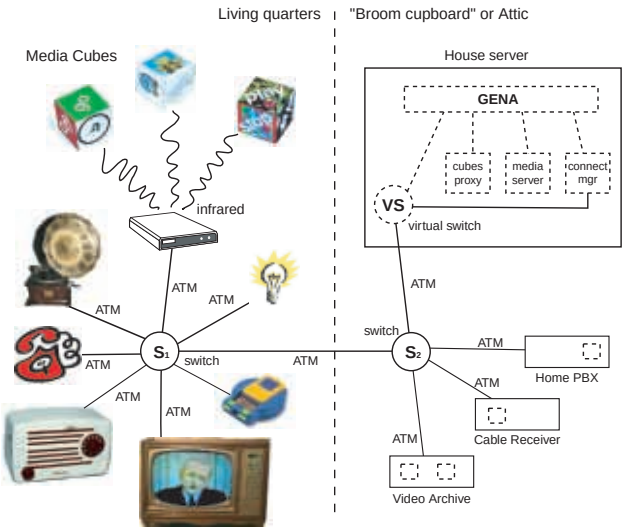

Figure 2 – AutoHAN architecture

An AutoHAN home network consists of a collection of devices, connected by a heterogeneous network. Some of these devices, such as a television, will be in the living space of the home. Others, such as a digital cable TV decoder, might only ever be accessed over the network, and as such may be placed out of the way, for example in a broom cupboard or attic. A third class of device consists of virtual devices – software components running somewhere on the network. As far as the user is concerned, these virtual devices behind the walls are no different to the invisible physical devices in the broom cupboard. In a typical setup, a "Home Server" would run a collection of virtual devices, in addition to key AutoHAN services. The home server need not be physically located in the living space.

The AutoHAN architecture is designed to be agnostic to underlying network technologies. In the current implementation, physical devices mainly communicate via ATM, while software processes communicate via IP. However, other network technologies can, and in several cases have, been integrated into AutoHAN. Examples include the infrared links to connect Media Cubes and other mobile devices, Ethernet, BlueTooth and IEEE 1394 (FireWire/i.Link).

4.1. GENA

AutoHAN devices normally communicate using the Universal Plug-and-Play (UPnP) Generic Event Notification Architecture (GENA) [17]. This is an extended form of Hypertext Transfer Protocal (HTTP). In cases where the underlying network protocol is not connection oriented, HTTP-U may be used. This, in addition to the simplicity of HTTP, allows event sources to be implemented in relatively modest hardware and software, making it well suited for embedding in consumer devices.

AutoHAN entities make a subscription request in order to receive certain types of events (the GENA Notification Type), setting up something akin to a web server. When an event of the specified type occurs, an HTTP "NOTIFY" request is sent to the subscriber, which responds with a confirmation that the event was received. Additional parameters, such as identifying exactly which button on a control panel was pressed, may be encoded in the GENA Notification Subtype, or in the message body.

4.2. The DHan registry service

A further key component of AutoHAN is the DHan registry, which enables devices and other entities to advertise their properties. The registry is accessed via standard HTTP; entities may use POST requests to add or update information about themselves, GET requests to look up information about other entities, and so on. DHan uses finite-time leases – when an entity registers information, it is given an expiry time, after which the information is removed if it has not been re-registered.

The DHan registry uses XML for storage and presentation of device properties. XML was chosen for several reasons; it allows for loose matching and future expansion, is supported by web browsers, can be transformed using XSL for presentation purposes, and allows the register to construct and present a hierarchy of devices. Objects in the hierarchy are identified via a point, a fully or partially qualified name in the style of the W3C's XLink and XPointer, or distinguished names in X.500.

4.3. User benefits of AutoHAN architecture

GENA and DHan, along with enabling low-level technologies, allow the AutoHAN home network to be largely self-configuring. When a device is added to the network, it sets up an event source, and registers itself with DHan. From then on, other entities may retrieve the details of the device via DHan, using an appropriate search string, and communicate with it via GENA. The use of leases enables the system to deal gracefully with network and power outages, and the failure or removal of individual devices.

Together, these components of the system ensure that the end user need not become a systems administrator in their own home. This type of self-configuration and graceful degradation is an essential feature of end-user programmable systems. They allow users to interact directly at the level of device functions rather than network infrastructure. This provides consistent remote control functionality across both physical appliances and virtual devices, as well as a platform for generic interaction between devices. User manipulation of the Media Cubes can define scripts which are archived in the registry and which, when executed, result in some new combination of subscription requests, events and channel configurations.

5. Programming paradigms for Media Cubes

The scripting capabilities of the AutoHAN Media Cubes support a wide range of abstract behaviour. We have taken great care not to make any unwarranted assumptions in the design of the Media Cube language. The physical form of the language is so unusual that it offers many potential styles of programming. There have been a few previous programming languages with physical tokens, such as the AlgoBlock system [18]. These systems have often been a relatively direct translation of some familiar programming style – AlgoBlock was based on the Logo language.

Rather than accept the constraint of any one familiar programming paradigm, we have considered two alternative paradigms for the Media Cubes language. One of these is based on a philosophy of ontological abstraction – the tokens of the language are designed to represent "natural categories" in the user's mental model of the AutoHAN operation. The other is based on a linguistic abstraction hypothesis, that abstractions from physical situations are naturally described in linguistic terms. This has resulted in two novel language paradigms, which are described separately below.

Both paradigms proceed from the assumption that Cubes can be associated with a physical appliance by placing the Cube next to the appliance, and that Cubes can also represent virtual devices, such as a media archive hosted on the home server. Both paradigms also allow for further generic Cube functions that provide abstraction capabilities independent of any specific device.

5.1. Ontological paradigm

Figure 3 – Cubes for ontological paradigm

The ontological paradigm for Media Cubes programming has been developed by identifying concepts for which there is a close correspondence between primitive remote control operations, appliance functions, capabilities of the AutoHAN architecture, and user skills. These represent a primitive ontology of home automation. Cubes are identified with ontological

types, which are indicated to the user by graphics on the Cube, as shown in figure 3. The faces of any given abstract Cube type correspond to the potential interpretations or interactions of that abstract type. To illustrate, we briefly describe four of the abstract types.

An *Event* Cube is a reference to a change of state in the home – either a sensor activation (such as a doorbell), an automated function (an alarm clock being activated) or any appliance function that can be specified by the press of a single button. The faces of the event Cube allow the user to define behaviour on transitions in either direction: "go"/"stop", or "on"/"off". "Go" and "on" being functionally identical, but labeled separately in order to help users reason about equivalence between events and processes.

A *Channel* Cube provides a reference to a media channel or stream allocated within the AutoHAN network. It can be used to associate a channel with a media source, and to direct data from that source to a media sink (an output device, an archive device or a reprocessing device). Media channel data types might include audio, video, image or text – these could either be identified with different Cube graphics, or all controlled by a single Cube. The latter option leads users to think about functionality of the network at a more abstract level. The faces of the channel Cube include channel content (interpreted as either source or sink, depending on the appliance the Cube is associated with), and a channel index, which allows specific temporal points (tracks, programme starts, phone messages) to be referred to.

An *Index* Cube selects content from a channel. Some channels carry indexing information such as time (of an answerphone message or speech archive), programme ID (from audio or video broadcast), or sender (of an email). The indexing Cube can be associated with a particular index value, and select content that matches the value. Sources of index values include time specifications, archive references, or current broadcasts.

An *Aggregate* Cube allows the user to refer to abstract collections rather than individual instances. Aggregates may be formed by associating a range of other Cubes to represent (for instance) a collection of events. An aggregate Cube need not be cube-shaped – it can be in the shape of a open box, so that the aggregation is indicated physically by placing other Cubes within the box.

The channel Cube potentially supports higher order functions that are accessible to AutoHAN users as they become more expert. The program scripts themselves can be represented as a channel, in which case they are also treated as first class objects in the Media Cubes language. Playing a script channel activates the script, which can be archived, indexed, or controlled by other events in the home. The user transition from direct specification of appliance functions to higher order programming thus follows the same abstraction gradient as the transition from direct operation of remote controls to basic scripting functions.

5.2. Linguistic paradigm

The second approach to the Media Cubes is based on linguistic abstraction. Here the Cubes represent words in a language. For example, one type of Cube has a single face labelled *Clone*. When this face is placed against another Cube face and activated, the "Clone" face takes on the identity and function of the other face, allowing users a "shorthand" for referring to physical (and virtual) objects. This is analogous to a variable binding in a conventional imperative programming language.

An extension of the *Clone* Cube is the *List* Cube. This has three active faces – "Add item", "Remove item" and "Contents". Each Cube has an associated list of items that it "contains", and the "Contents" face aliases this list. A type system is to be used to ensure that the contents of the list "make sense". As with all scripts, the list is not stored in the Cube itself, but in a proxy object residing in a server. The Cubes themselves are identical and interchangeable, aside from their labels and a unique identifier that used to discriminate between their infrared signals. A Cube of this type is similar to an object in a language such as Java or Smalltalk – the Cube has an identity, associated with which is a value and operations.

Standard consumer remote controls may be integrated with the Media Cubes language by adding induction coils to them, or by exploiting the directionality of their infrared signal. This allows users to create abstractions of the functions associated with regular remote controls. For example, when we point the television IR controller at the TV and press '1' it turns on the TV on BBC-1. If instead we point the same controller at a 'time' Cube, then pressing the '1' button on the IR controller defines a one-time program to turn on the TV on BBC-1 at that time tomorrow.

One question that the linguistic approach highlights is that of the computational power required by the language; in particular, should the language be Turing powerful? Although current home control interfaces are certainly not, we decided to experiment with the inclusion of higher order operations having the potential to support powerful abstractions. This allows us to investigate both the new applications that such a language allows, and any potential pitfalls that it brings about.

Another factor in the language design is the degree to which the arrangement of Cubes may be dynamic. The design outlined above is highly dynamic, in that the associations between faces that define the arrangement may be created over a period of time. A static arrangement, in which the entire program is laid out at once, would have the advantage of ensuring that the program as a whole was visible. However, in this

type of system, the complexity of the program is limited by both physical constraints (eg, not having enough space to lay out the program you desire, or wanting to place several Cubes next to a single face of another), and by the number of Cubes available. The ontological approach would seem to favour a static arrangement, and the linguistic approach a dynamic one, but in both cases it is likely that the optimal solution lies somewhere between those two extremes.

10. Conclusions

We have created an environment that supports a new style of end-user programming for domestic contexts. This has required extensive development of prototype appliances, network infrastructure, and a prototype programming tool called Media Cubes. This combination offers an exciting platform for research into the creation of usable abstraction facilities for the home that will be accessible to a wide range of domestic users.

10.1. Further work

At the time of writing, AutoHAN and the Media Cubes are operational. In keeping with the psychological concerns of the project, we do not consider implementation to be the end of the project. The next phase is to carry out extensive user evaluation, especially of the two alternative paradigms for the Media Cubes language.

11. Acknowledgements

Daniel Gordon designed the Media Cube prototypes, and they were constructed by Dick Kimpton. The AutoHAN repository and networking protocols were developed by Umar Saif and Daniel Gordon. The technical implementation of the AutoHAN project is led by David Greaves. Alan Blackwell's research is funded by the Engineering and Physical Sciences Research Council under EPSRC grant GR/M16924 "New paradigms for visual interaction". Rob Hague's research is funded by the EPSRC and AT&T Laboratories Cambridge.

12. References

[1] J. Waldo, *JiniTM technology architectural overview*, http://www.sun.com/jini/whitepapers/architecture.html, Sun Microsystems, 1999.

[2] Microsoft Corporation, *Universal Plug and Play device architecture* http://www.upnp.org/download/ UPnPDA10_20000613.htm, 2000.

[3] U. Hansmann, L. Merk, M.S. Nicklous & T. Stober, *Pervasive Computing Handbook*, Springer-Verlag, 2001.

[4] B. Brumitt, "Easy Living", seminar presentation at Microsoft Research, Cambridge, 10 December 1999.

[5] A.F. Blackwell, "Pictorial representation and metaphor in visual language design". To appear in *Journal of Visual Languages and Computing*, June 2001.

[6] A.F. Blackwell & T.R.G. Green, "Does metaphor increase visual language usability?" In *Proc. IEEE Symp. on Visual Languages*, 1999, pp. 246-253.

[7] T.R.G. Green & M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework". *Journal of Visual Languages and Computing* 7, 1996, 131-174.

[8] T.R.G. Green & A.F. Blackwell, Design for usability using Cognitive Dimensions. Tutorial at BCS conf. on Human Computer Interaction, 1998 http://www.cl.cam.ac.uk/~afb21/ CognitiveDimensions/CDtutorial.pdf

[9] A.F. Blackwell, T.R.G. Green & R.L. Hewson, "Product design to support user abstractions". Paper submitted in June 2000 to special issue of *ACM ToCHI* on "The New Usability".

[10] D. Kahneman & A. Tversky, "Prospect theory: an analysis of decision under risk". *Econometrica* 47(2), 1979, pp. 263-291.

[11] A.F. Blackwell & T.R.G. Green, "Investment of attention as an analytic approach to Cognitive Dimensions". In *Proc. 11th Ann. Workshop of the Psychology of Programming Interest Group*, 1999, pp. 24-35.

[12] *The Visual End User*: unpublished workshop papers on visual languages for end-user and domain-specific programming. September 10, 2000, Seattle, WA, USA.

[13] A.F. Blackwell, "See What You Need: Helping end users to build abstractions". Paper submitted in December 2000 to *Journal of Visual Languages and Computing*, special issue on End-User and Domain-Specific Programming.

[14] A.F. Blackwell, "SWYN: A visual representation for regular expressions". In H. Lieberman (Ed.), *Your wish is my command: Giving users the power to instruct their software.* Morgan Kauffman, 2001, pp. 245-270.

[15] A. Repenning & J. Ambach, "Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing". In *Proc. IEEE Symp. on Visual Languages*. Boulder, CO. 1996, pp. 102-109.

[16] U. Saif, D. Gordon, D. Greaves. *IEEE Internet Computing* February 2001, pp. 54-63

[17] J. Cohen, S. Aggarwal, and Y.Y. Goland, *General Event Notification Architecture Base: Client to Arbiter, work in progress*. Internet draft, (expires Apr. 2000), available at http://www.upnp.org/draft-cohen-gena-client-01.txt.

[18] H. Suzuki & H. Kato, "Interaction-level support for collaborative learning: Algoblock – an open programming language". In *Proc. Comp. Supported Collaborative Learning '95*. Lawrence Erlbaum, 1995

[19] H. Ishii & B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms". In *Proc. CHI '97*, 1997