Designing a Programming Language for Home Automation

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

Keywords: POP-I.C. *domestic automation* POP-II.A. *end-user*
POP-III.B. *ML family languages* POP-III.C. *tangible languages*
POP-VI.C. *likely future developments*

Abstract

The AutoHAN project at the Cambridge Computer Laboratory is developing a range of technologies related to next-generation home networking and automation. Several aspects of the project involve the development of programming languages suitable for use in the home. Languages of this sort will clearly have a significant impact on the usability of domestic electronic devices, and greatly broaden the range of users who might benefit from psychology of programming research. As yet, the design of the AutoHAN languages is not complete. This paper therefore describes the design process and design criteria that have been applied so far. This is as a basis for discussion at this workshop, and for planning of further psychology of programming research related to the AutoHAN programme.

Home Automation Background

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. Waldo/Sun Microsystem 1999, Microsoft Corporation 2000) by which appliances can communicate with control devices and with each other.

These trends will soon bring significant challenges to the psychology of programming community. PPIG researchers have occasionally wondered in the past whether the proverbial impossibility of programming a video cassette recorder should be a concern of ours. A few people have investigated the area, but not many. Thomas Green, Alan Blackwell and Rachel Hewson have given some thought to the programming of central heating systems (Blackwell, Green & Hewson submitted), as have Anna Cox and Richard Young (2000). Harold Thimbleby (1993) has analysed the programming interfaces of microwave ovens, and even VCRs (Thimbleby 1991). But even though these results might bring psychological or usability insights, they are seldom considered to be directly relevant to programming language design. "Programming" a VCR is not real programming in the eyes of most computer scientists. But if you want to instruct your VCR to start recording from the front door security camera for 5 minutes after the time that someone presses the front doorbell, this seems a lot more like real programming.

So the significance of home networking 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? Perhaps the manufacturer will build in all the required functionality to both devices. But would you trust your VCR manufacturer to do this? Perhaps manufacturers will just keep themselves to themselves, and won't be tempted to mess with other devices in your house. Would you trust them to do that? On balance, it does seem that homeowners will find themselves with the potential, and maybe the inclination, to define complicated behaviour in their own homes.

The AutoHAN project

This problem is the main concern of the AutoHAN project in the Cambridge Computer Laboratory. The Home Area Network (HAN) project continues some years of research, developing hardware and software tools suitable for home networking. A demonstration house has been wired up with the original "Warren" system, which can be used to network radio, CD players, video recorders and

telephones to speakers, displays and interaction devices all over the house. All media and control signals are carried using the high-speed ATM protocol, implemented in low-cost custom chips. The Warren system has been described elsewhere (Greaves 1997).

AutoHAN is a follow-on project that aims to give homeowners the ability to interact with their home area network and define its behaviour. This involves many complex pieces of software – a home registry, an architecture for mobile software, event architectures, media, authorisation, validation and subscription services. Those aspects of the project have been described elsewhere (Saif, Gordon & Greaves 2001). This paper is concerned with the development of an interface to these various facilities for the home programmer.

Other related projects

As we have said above, there are many research groups, companies and consortia who are racing to develop home networking technology. Some of those have had to consider the issues of how a user will specify behaviour. The IBM pervasive computing group, in their recent textbook on pervasive computing (Hansmann et al. 2001), report the findings of their research on usability: such devices should be both “convenient” and “intuitive” (Hansmann et al., page 23). 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 (Brumitt 1999). Our goal in AutoHAN is to improve on these efforts.

Overview of AutoHAN programming

Researchers on the AutoHAN project are currently working on several different programming languages. These are aimed at different types of users, and some of them might only ever be used by computer science researchers (members of our project team). The two main languages (which only have provisional names at this stage) are referred to as “Iota.HAN” and “Media Cubes”. Both of these languages are only partially specified at present. It may seem strange that a research paper is being written about a partially specified language, but this is completely intentional, as described below.

Iota.HAN

The Iota.HAN language is intended to be a systems programming language for AutoHAN. Some of the design objectives for this language are: that it should be suitable for manipulating the home registry (with data stored in XML format); that it should be suitable for processing and responding to events; that it should be possible to update the set of programs while the network is running; and that it should be possible to carry out some automatic analysis of programs written in the language to check for correctness, or for violation of secure constraints (e.g. do not allow any appliance controlled by teenage residents to debit the household account without authorisation). The designers of Iota.HAN have more general interests in language design, which they are pursuing under the name “Iota”. Iota.HAN is a customised version of Iota (and also the first implemented version of Iota) dedicated to AutoHAN.

Media Cubes language

The Media Cubes language is intended for “end-user” programming in the home. It has very little resemblance to a conventional programming language. The syntactic elements are small physical blocks (cubes about 5cm on each side) containing a range of sensing and communication interfaces, including infrared transceivers for communication with the home network, and induction coils that sense the proximity of other cubes. Programs are created in the Media Cubes language by placing cubes next to each other, and instructing the system to record that arrangement somewhere within the network.

Other languages

The AutoHAN project has also involved the development of a number of other programming languages. A predecessor to Iota.HAN was the Cambridge Event Language (CEL), which was originally developed in another research group of the Computer Laboratory. It was developed for use in a real-time event database, and could be used to describe and retrieve generalised events from the database. In AutoHAN, CEL would have been used to define policy – what other actions should be taken when given event combinations occurred. One student worked for two years on the adaptation of CEL to AutoHAN, but this work has been discontinued, as a result of the student leaving the Laboratory.

CRISP ("C*** LISP") is a command language with LISP-like syntax, designed as part of an AutoHAN event server. It was written quickly, largely as an exercise, and as such, little effort was put into the language design. The intended use of the language was to allow system administrators to alter the configuration of a server at runtime. Work on CRISP resulted in some ideas with a wider application, such as having a standard XML representation of the language.

Relevance to PPIG

At present the languages being developed on the AutoHAN project are only research tools. There may never be real users struggling to program a toaster in these languages. But research tools have a funny way of turning into commercial products. It may not turn out be our particular languages that become commercial products, but somewhere in the world, a research team is developing a language for home automation that will be the core of an actual future standard that we will all have to live with in our houses.

This gives an unusual opportunity for PPIG research. PPIG researchers spend a lot of time looking at the standard languages of today (C++, for example), and asking each other "why did the designers do this?" In the case of C++, the author of the language did publish a book justifying his decisions (Stroustrop 1994), but it might be sensible to take a sceptical view of the accuracy of his recall – especially where there have been mistakes or backtracking. In the AutoHAN project, there is a chance to observe language designers in their native environment, designing languages that, even if they do not become international standards, are "like" the languages that will become the standards. They are "like" in the sense that they are being developed in a similar environment, using similar tools, in the background of similar previous research. This observation can therefore give us some insight into the very early stages of designing a language for usability.

The design of the Media Cubes language

Language style and origins

The Media Cubes language is intended for use solely as an end-user programming language in the home. It was conceived as an alternative to an earlier proposal in which the HAN would be programmed by an infrared wand incorporating voice-recognition facilities. The drawback of that proposal was that the user would never be able to see the program under construction – it would consist solely of a sequence of transient gestures and spoken commands. The Media Cubes proposal was for a relatively large number of physical control elements, each communicating with the network, and each corresponding to some class of program functionality – e.g. representing the occurrence of an event, the state of a media channel, a time of day etc. These cubes could be arranged on a surface (or in the user's hands) so that the ordering of cubes and contact between specific faces represented the required program. The program under construction would be visible, and could be manipulated directly without use of a computer. This concept is clearly related to previous proposals for physical programming interfaces where flowcharts are constructed by assembling elements such as Lego bricks (Suzuki & Kato 1995). In the Media Cubes language, however, the program elements are not those of a conventional programming language. They are intended to correspond to the conceptual operations performed with remote controls – and hence to familiar physical renderings of abstract appliance functions.

Overview of current design

Figure 1 – Mockup showing Media Cube functions

The current conception of the language design centres on the notion of placing the faces of two cubes adjacent to each other, and pressing the buttons on the cubes to explicitly indicate a relationship between the two. The faces of the cubes are labelled with operations or values. It is expected that some of these values will be transient, for example a face that has taken on the value of a particular time, while others will be fixed, such as a “Play” operation for media streams. Figure 1 shows a physical mockup of the Media Cubes constructed to demonstrate this.

Each cube may also have an associated action that is triggered by pressing the button while there are no adjacent cubes. Used in this way, the cube acts like a simplified version of a conventional remote control.

As well as referring to other cubes, the Media Cubes language can be extended into the environment by fixing coils onto devices. This allows a device to be specified directly, by physically placing a cube next to it, as opposed to indirectly, by a specification such as “tv004” or “that wide-screen TV in the corner of the lounge”, or a selection from a list or map. 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.

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”. Note that 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.

The “cubes” used for programming need not be uniform. A “time” cube, for example, may take the form of a clock face or digital display on which one can set a time, and have a face that takes on the value of the displayed time, and another that, when programmed with an action from another cube, performs that action at the specified time.

Standard consumer remote controls may be integrated with the Media Cubes language via adding coils, or exploiting the directionality of their infrared signal. This allows users to create abstractions of the functions normally associated with the controllers. For example, 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 the `time' cube described above, 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.

Language design activities

The Media Cubes language is still at an early stage, and as such much of the design work done to date is exploratory in nature. As there are few similar languages, ideas have been adapted from other

domains, such as Visual Programming. The novelty of the language, coupled with the relative immaturity of the hardware, has also meant that, in parallel with the language design, many technical aspects of the language implementation have had to be addressed.

One question that was raised early on in the language design process was 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 placed several cubes next to a single face of another), and by the number of cubes available.

Figure 2 – working prototype Media Cubes (with pencil for scale)

To try out and expand on these ideas, a small number of prototype Media Cubes have been manufactured. Two of these are shown in Figure 2. Each cube has a copper coil on four sides. Induction in these coils allows a cube to detect when another cube is brought within close proximity to one of its faces. The cubes also have a single button and can communicate to a base station via an infrared link. All functions of the cube are controlled by a PIC micro-controller, which includes power management functions intended to achieve a battery life of about one year. The power management functions are controlled by a motion sensor, so that they fall into quiescent mode after a few minutes of immobility, and are immediately reactivated when picked up or moved by a user. The internal circuitry of a prototype cube is illustrated in Figure 3.

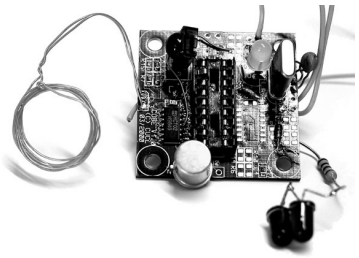

Figure 3 – internal circuitry of a prototype Media Cube

Observed design criteria

The Media Cubes language assumes the benefits of physical manipulation as a programming technique. Some programming languages have previously been described as “tactile” (Repenning &

Ambach 1996), but in fact still consist of visual representations on a computer screen. The "Tangible Media" group at MIT has created a number of systems integrating "physical icons" into the interaction environment – these are regarded as extending the advantages of direct manipulation (Resnick et al. 1995). The "Lego" ideal of programming as the assembly of interchangeable components has occasionally been taken literally (Suzuki & Kato 1995). In the Media Cubes language, however, the main benefit of the language is in the way that users' previous experience of remote controls can be exploited to make the cubes both familiar and logically extensible. Nardi (1993) recommends that end-user programming should exploit existing visual formalisms, extending the formalism with computational capabilities in the same way as was achieved with the spreadsheet. In the Media Cubes language, we treat the familiar domestic remote control as a "physical formalism" that can be extended in an analogous way.

This has resulted in some rather different design criteria when compared to conventional programming languages. The syntactic elements, rather than being derived from other programming languages as in the Algoblock system (Suzuki & Kato 1995), have been chosen to correspond to familiar physical formalisms from domestic devices. They therefore include play and pause operations, on/off operations and entities that represent broadcast channels, track indexes, clock settings and telephone numbers. The formal definition of these functions is still under development.

Association of the abstract functions with the behaviour or state of actual appliances in the house will be achieved by a physical action: the cube will be placed against an appliance in order to associate it temporarily with the behaviour of that appliance.

The greatest drawback of the Media Cubes proposal is that it is a write-only system at present. Cubes can be arranged by the user, and the resulting program recorded in the HAN, but if the cubes are then used to build another program, the original is no longer available for inspection. It is clear that HAN users will want to inspect and modify their programs. At present, we envisage that completed programs might be accessible using a program cube type, and that this can be used to display a previously recorded script on some suitable display device – a TV screen, for example. It is clear that many of the cognitive dimensions of notations will identify challenging usability issues associated with this proposal.

Future objectives

It seems that the Media Cubes language may be more generally applicable to defining the interaction of networked appliances beyond the AutoHAN project. Low cost construction techniques have led recently to a flurry of small physical devices used for network interaction. The Xerox "Satchel" system, for example, is used as a token for specifying the transfer of data between systems and to peripherals (Lamming, Eldridge et al. 2000). Smart card identification tokens are being used to transfer user sessions between public terminals. The Bluetooth wireless protocol will make such applications both economical and more robust. Despite these factors, this interaction style has not yet been applied to the domestic context. The resemblance of the cubes to remote controls should make them especially appealing in that environment. This commercial potential has led to the precaution of applying for patent protection (Blackwell 2001), in case of future commercial spinoffs from the project.

On the technical side, the fact that the Media Cubes have been specified from the perspective of user functionality has left a substantial gulf between the behaviour of the Media Cubes and the underlying network applications. Our current objective is that programs created with the cubes should be compiled into scripts in Iota.HAN, but it is not yet clear how this will be achieved.

We also intend to conduct experimental evaluations of the usability of the Cube language. Our basic approach will be to familiarise users with the concrete functionality of the Media Cubes, when used directly as remote control devices. Once users have become accustomed to the use of cubes for concrete manipulation of system state, we believe that the transition to abstract programming using the cubes as representatives of state will be relatively straightforward. This is similar to the

psychological claims made for other “tangible” programming systems (e.g. Repenning & Ambach 1996), but in our case the “tangible” language components are in fact tangible physical objects.

The design of the Iota.HAN language

Language style and origins

The design of Iota is being led by Gavin Bierman (GB) and Peter Sewell (PS), with assistance from an undergraduate student who is implementing the first compiler. It is based on earlier work by GB, investigating possible programming tools for WAP phones, and by PS, on languages for concurrent programming. Although their earlier projects were independent, both had been investigating mechanisms for processing XML (Extensible Markup Language) data – for WAP page markup (GB), and as a representation for home automation (PS) after a direct suggestion by the AutoHAN project leader. They therefore started from a shared interest in XML as a first-class data type, with further interests in small-footprint languages and concurrency. Both had worked extensively with functional languages such as ML and other *impure, statically typed, call by value*, languages in the past. They considered this paradigm a natural choice for a new language design. ML is actually a family of languages (the best known being Standard ML). As a shorthand term for all impure, statically typed, call by value languages, we will use the term “ML-like functional languages” (MLLFL) in this paper

Overview of current design

The current design of the language is well defined, in that its syntax, type system and operational semantics are formally specified in the manner of Milner et al (1997). However, at this stage all is subject to change. In particular, the language designers stress that the choice of primitives, and also concrete syntax, can and should be refined according to observations of the language in use.

```
fun GetAge <person age=x>
             *:content list
           </person>          => x
         | *:MU               => “-1”;
```

Figure 4 – An example function definition in Iota.HAN

As mentioned above, Iota is a functional language in the style of MLLFL, and the syntax reflects this. XML is handled by adding a builtin markup type (`MU`) and extending the MLLFL pattern matching language accordingly. For example, the function `GetAge` defined in the example of Figure 4. matches a “`person`” tag and binds the name “`x`” to the value of its “`age`” attribute, then returns the value bound to “`x`”. Note the pattern “`*: type`” matches any value of the correct type, but makes no bindings. This arrangement is used in the first pattern to ignore the contents of the tag, and in the second case to match any markup that does not match the first pattern and return a default result. In general, binders in Iota must be explicitly typed, although in the case of some patterns and literal values, the type is implicit.

Concurrency in Iota is expressed using Pi-calculus style channels (Milner, Parrow and Walker 1992). This notation was chosen to give a simple yet flexible way of expressing concurrency. It also has the advantage of being based on a well understood formalism, and is flexible enough to be used as a basis for higher-level concurrency constructs. The terse concrete syntax is similar to that of Pict, a “language based directly on the Pi-Calculus” (Pierce and Turner 1997), which among other features indicates input and output channel identifiers using the concurrent language convention of “`!`” (output) and “`?`” (input).

At present, Iota includes polymorphism through subtyping, but lacks parametric polymorphism as found in many functional languages. This is due to a desire to keep the language simple and compact, while accommodating a more flexible typing system used for markup. A more complex type system,

which would provide features such as modules and data hiding, may be added at a later date, but such decisions will have to be weighed against the requirement for a small, simple language.

Language design activities

This section summarises results from research interviews conducted with the designers, in which they were asked to provide a "snapshot" of the design rationale for Iota. The interviews were conducted before the first internal document on Iota had been completed, and after they had been working on the project for about six months. GB provided most of the material reported – PS was sceptical that such information would be valuable without detailed discussion of the design space. In the following presentation of material from the interviews, key terms are italicised – these key terms were repeatedly discussed in the interviews and subsequent meetings.

The starting point for the design was a collection of current material from the AutoHAN project. These included partial specifications from the UPnP (Universal Plug & Play) home networking standard, and scenarios of AutoHAN execution. These documents included samples of XML that should be transformed or generated in a home network. The designers observed conflict between the *concrete syntax* of the two languages. XML is "only a transfer language, designed by non-academics", while ML is a "most beautiful language". In combining the two languages, they were faced with the choice of either extending XML with escaped chunks of Iota, of handling quoted XML data values within Iota code, or integrating the two without escaping. They decided on the latter.

A second major concern was with type processing in the language. MLLFL are *strongly typed* languages – they were designed by researchers with a central concern for programming language semantics. GB and PS themselves are active researchers in this area. Unfortunately type information in XML is not well defined. It is presently loosely specified via Document Type Declarations (DTDs), and the W3C consortium is considering proposals for "XML Schema", a more expressive definition language. However, this is just one of many competing proposals for an XML type system, and no decision is likely before the first implementation of Iota is complete. The decision was made to use a crude type system for XML to avoid the subtleties of systems like Xduce (Hosoya & Pierce 2000) and XMLamda in Iota – all XML data will belong to a single type. This is likely to cause severe difficulties in automated reasoning about program correctness. The only way to avoid this is for Iota programmers to adopt a style with assertions and "catch-all" expressions.

Apart from discussions between the designers, the first group discussion of the Iota syntax was in an AutoHAN weekly progress meeting – about six weeks before these interviews. This meeting revealed that project members had different preconceptions about the nature of XML. The person who had done most work on registry design seemed to expect database-like functionality when processing XML. As a result of this meeting, a *comprehension* construct was added to the language. The designers later expressed doubt of the value of the original meeting, as the public situation meant only unrepresentative samples of functionality could be discussed.

Shortly after these interviews, the designers prepared a preliminary written presentation of the design, including comments on design rationale that may have been prompted by the interviews. This was distributed to the project team. The most recent design activities have been a meeting of the AutoHAN project team at which this document was discussed, and a further discussion with the designers.

Observed design criteria

The preliminary specification document with the Iota.HAN design rationale describes a number of desirable language properties, and also some tensions between conflicting properties that may be desirable, but are incompatible. In an analysis using Cognitive Dimensions of Notations (CDs) (Green & Petre 1996, Green & Blackwell 1998), these conflicting properties would be described as trade-offs. In fact it is interesting to compare the various properties described as desirable in this document to the language properties described by CDs. CDs were originally proposed as a vocabulary that can be used by language designers to discuss the usability decisions they make. The

Iota.HAN document is a valuable sample of the actual informal vocabulary that is used by language designers in early design phases. It is presented here for later comparison with the CDs vocabulary. The key elements of this vocabulary are italicised below.

An overall objective for Iota has been to achieve *well-defined* and *clean* semantics. The resulting execution behaviour should be *predictable*, and *precise*. This can best be achieved by making the language *small* and *simple*, but there is a tension between simplicity and *expressiveness*. The concurrency features should be both expressive and *lightweight*. The type system for XML data is not very expressive, as this would be too *complex*. Instead a *loose* and *primitive* type system has been chosen (though a more expressive one may be retrofitted).

The language is intended to provide a *clean* abstraction layer. This includes a *lightweight* base language at the right level of *expressiveness*, as well as library channels that are *tuned* to HAN communication requirements. The implementation must also have a *small footprint*.

Some of the desirable attributes of the design are that it provides *first-class* integration of XML data, *higher-order* functions, *rich* pattern matching, exception handling, asynchronous communication primitives that are *lightweight* but *expressive* and a *simple, straightforward* type system. The type system will enable a *simple programming style*, which doesn't require *complicated* type annotations, so code can be *developed quickly*. The Iota type system is *flexible*, but *weaker* than MLLFL. The language also achieves reduced *verbosity* compared to XML processing in other languages by providing *lighter* syntax. *Heavy* syntax has been avoided by removing unnecessary escaping.

Premature concern with *concrete syntax* is generally viewed as a distraction in the design process. Some of the first discussions between the designers did relate to concrete syntax (embedding XML in Iota, as opposed to vice versa), but they were keen to defer final decisions. The syntax definition does refer to some *surface syntax* "magic", but this is not widespread.

Many of the above aspects of the language are related to its eventual usability, but the designers did not specifically mention the intended class of users either in the specification document or in interviews eliciting their design criteria. When reviewing the draft specification, the AutoHAN project leader presented two specific classes of user that were expected to use Iota.HAN: design engineers working for appliance manufacturers and tradesmen customising home network installations. This is clearly a challenging requirement – in the preparation of this paper, it emerged that the Iota designers thought it had been a joke.

It seems that many of the issues investigated in psychology of programming research have not been of great value in making the design decisions for Iota.HAN. For example, the desire for simple informal and formal reasoning about the behaviour of Iota.HAN programs, and consquent early choice of MLLFL as the most appropriate basis for the language has forced a bias towards recursive (as opposed to iterative) control paradigms. Good MLLFL style relies on the programmer having some facility with recursive programming, and any attempt to change this would discard many benefits of functional programming. ML has been described by Ousterhout as "a language for people with excess IQ points" (Haemer 1995) perhaps explaining its popularity in Cambridge as a "levelling" device for undergraduates (Stajano 2000). We do not expect this to observe this in the case of domestic tradesmen.

Future objectives

The development of Iota has reached a stage such that members of the AutoHAN project can write reasonably-sized programs in the language. This is expected to lead to refinement of the language design.

As mentioned above, one area that may see significant changes is the type system. The present implementation supports limited type inference, and has parametric polymorphism internally. Depending on the observations of initial users of the language, these features may be removed, or expanded upon.

The possibility of adding an XML representation of Iota is being considered. This would allow alternative, interchangeable concrete syntaxes to be produced without altering the underlying language or its implementation, hence providing a tool with which to experiment with a wide variety of end-user languages. As well as simply providing alternative textual forms, diagrammatic forms, perhaps similar to that described by Erwig (2000), could be tested. Also, as all of these higher-level languages share a basis in the XML representation, programs could be viewed in any of these languages without modification.

Summary

The AutoHAN project is proceeding in a context that is quite typical of the research origins of many widely used technologies. Commercial issues, technical constraints, and standardisation efforts are continually influencing the design decisions being taken. Yet the project team are also pursuing research objectives that are independent of these considerations. The Media Cubes language is a research platform for novel approaches to language usability, while the Iota.HAN language represents a new approach to the deployment of functional programming languages in an embedded real-time control environment.

These differing research objectives have provided us with an opportunity to present the contrast between the design criteria that are applied to programming languages at different levels of system architecture. Neither language design is complete, but the design criteria are quite clearly elaborated at this stage of the project.

As one of the long-term goals of psychology of programming research is to influence the criteria applied to language design, this observational study provides a broad base, from two very different perspectives, of the domain of application for that research.

It remains to be seen whether either of these languages will have any significant influence on future developments in home area networking, but we contend that the organisational factors that led to their design are quite typical of other research groups. Whatever languages do become applied in domestic situations, they are likely to have arisen from similar research or development environments. If psychology of programming research is to have an influence on the next generation of ubiquitous programming languages, it should address itself in a recognisable manner to the factors that have been of concern here.

Acknowledgements

We would like to thank Gavin Bierman and Peter Sewell for their patience in being treated as subjects of study, and for their assistance in reviewing the way we have presented (and occasionally misunderstood) their work on Iota. Daniel Gordon designed the Media Cube prototypes, and they were constructed by Dick Kimpton. 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.

References

Blackwell, A.F. (2001) *Remote control system for defining interaction between electronic devices*. New British patent application No. 0001921.6

Blackwell, A.F., Green, T.R.G. & Hewson, R.L. (submitted) Product Design to Support User Abstractions. Submitted in June 2000 to special issue of *ACM ToCHI* on "The New Usability".

Brumitt, B. (1999). *Easy Living*. Seminar presentation at Microsoft Research, Cambridge. 10 December 1999.

Cox, A.L. & Young, R.M. (2000). Device-Oriented and Task-Oriented Exploratory Learning of Interactive Devices. In N. Taatgen & J. Aasman (eds.), *Proceedings of the Third International Conference on Cognitive Modeling*. Veenendaal, The Netherlands: Universal Press, pp. 70-77.

Erwig, M. (2000). A Visual Language for XML. *IEEE Symposium on Visual Languages 2000*.

Lamming, M., Eldridge, M., Flynn, M., Jones, C. & Pendlebury, D. (2000). Satchel: providing access to any document, any time, anywhere. *ACM Transactions on Computer-Human Interaction*, 7(3), 322-352.

Greaves, D. (1997). ATM in the Home and the Home Area Network. Presentation at *IEE Colloquium on ATM in Professional and Consumer Applications*. http://www.cl.cam.ac.uk/Research/SRG/netos/han/docs/

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

Green, T.R.G. & Blackwell, A.F. (1998). Design for usability using Cognitive Dimensions. Tutorial session at *British Computer Society conference on Human Computer Interaction HCI'98*. Also online, available from: http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf

Haemer, J. (1995) Very High Level Languages Symposium Report. *login* 20(1), 5-10. (Report from Usenix symposium Santa Fe, Oct. 1994)

Hansmann, U., Merk, L., Nicklous, M.S. & Stober, T. (2001). *Pervasive Computing Handbook*. Berlin: Springer-Verlag.

Hosoya, H. & Pierce, B.C. (2000). XDuce: A typed XML processing language. In *Proceedings 3rd International Workshop on the Web and Databases (WebDB2000)*.

Microsoft Corporation (2000). *Universal Plug and Play Device Architecture*. Available from http://www.upnp.org/download/UPnPDA10_20000613.htm

Milner, R., Parrow, J. & Walker, D. (1992) A Calculus of mobile processes, parts I and II. *Information and Communication*, 100(1), 1-77

Milner, R., Tofte, M. & Harper, R. (1997). *The Definition of Standard ML*. Cambridge, Mass: MIT Press.

Nardi, B.A. (1993). *A small matter of programming: Perspectives on end user computing.* Cambridge, MA: MIT Press.

Pierce, B. & Turner, D. (1997) *A programming language based on the pi-calculus*. Technical report, Computer Science Department, Indiana University.

Repenning, A. & Ambach, J. (1996). Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*. Boulder, CO. pp. 102-109.

Resnick, M., Martin, F., Sargent, R. & Silverman, B. (1996). Programmable bricks: Toys to think with. *IBM Systems Journal*, 35(3&4), 443-452.

Saif, U., Gordon, D. & Greaves, D. (2001). Internet access to a home area network. *IEEE Internet Computing* Jan/Feb, 54-63.

Stajano, F. (2000). Python in Education: Raising a Generation of Native Speakers. In *Proceedings of the 8th International Python Conference*, Washington DC, 24-27 January 2000.

Stroustrup, B. (1994). *The Design and Evolution of C++*. Murray Hill, NJ: Addison-Wesley

Suzuki, H. & Kato, H. (1995). Interaction-level support for collaborative learning: Algoblock – an open programming language. In *Proceedings of Computer Supported Collaborative Learning '95*. Lawrence Erlbaum.

Thimbleby, H. (1991). Can Anyone Work the Video? *New Scientist*, 129(1757), 48-51.

Thimbleby, H. (1993). Frustrations of a Pushbutton World. In *Yearbook of Science and the Future*. Chicago, Il.: Encyclopaedia Britannica.

Waldo, J. / Sun Microsystems (1999). *Jini™ Technology Architectural Overview*. Online publication, available from: http://www.sun.com/jini/whitepapers/architecture.html