Towards Ubiquitous End-User Programming

Rob Hague **Peter Robinson** **Alan Blackwell**
University of Cambridge Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD UK
{Rob.Hague, Peter.Robinson, Alan.Blackwell}@cl.cam.ac.uk

INTRODUCTION

We believe that end-user programming capabilities are an essential part of any flexible ubiquitous computing system. When these are well designed, and tightly integrated with the system as a whole, they allow users to add functionality that was not, and in many cases, could not have been, anticipated by the system's designers. This enables users to benefit fully from the possibilities ubiquitous computing offers. However, End-User Programming in a ubiquitous computing context faces several novel issues, in particular the communication channels available and the diversity of the user population.

We have taken as the domain for our research the domestic environment. There are already a range of programmable microprocessor controlled devices that are routinely found in the home, ranging from alarm clocks, security systems and boiler controls to VCRs and personal video recorders such as TiVo™. Several of these devices already pose a notorious usability problem for large segments of the population [1]. As home appliances start to interact with each other, the complexities of end user programming and customisation will become far more severe. Home networking systems are already becoming a widespread site of ubiquitous computing, both in research prototypes [2], and (in more limited form) in existing systems such as X10.

Is it possible that home-owners will ever be able to configure and customise interaction between the appliances in their homes? This is a critical question for the acceptance of ubiquitous computing. If the combined functionality of many appliances is no more powerful than that of the individual appliances purchased separately, then ubiquitous computing will not have any significant impact on regular lifestyles.

We have applied recent theoretical approaches in end-user programming [3] to the problem of domestic automation. Unlike end-user programming in the business context, where programming is done by "power-users" with respectable (if incomplete) technical knowledge, programming in the home can be done by people with a very wide range of abilities. The end user programming languages in products such as Excel already present a serious design challenge in supporting both casual users and serious developers [5]. In the domestic environment, we recommend an approach in which a range of programming paradigms are made available via a common programming architecture to support different modes of interaction with the underlying ubiquitous computing architecture. Not only will this support a range of user abilities, but also different programming tasks, as psychology of programming research has demonstrated that no language can be best for all applications - different notations give better support for different programmer activities (for example, creating a new program, modifying an existing program [4]). Hence, the system should allow a single program to be represented in a variety of notations for different users and different tasks.

LINGUA FRANCA - SCRIPTING IN MANY LANGUAGES

In order to create a system in which a user may manipulate a single program via multiple notations, we have designed *Lingua Franca*, a common XML-based intermediate form for scripting languages. Using this intermediate form has several advantages. For example, automated enforcement of policies that limit the action of scripts is of particular importance for end-user programming in domestic ubiquitous computing. Both home owners and authorities are likely to be concerned that end-user programs should not inadvertently or maliciously bypass fire alarms, security systems, or payment mechanisms. In order to achieve these safety provisions, the system must be able to reason about the behaviour of new programs as they are created, in order to assess whether they conflict with existing policies. The common representation allows a common enforcement mechanism across languages.

Lingua Franca goes beyond conventional multiple-language systems in it's support for translations *between* source languages (as opposed to simply translating multiple source languages into the same form of object code). Various source languages in *Lingua Franca* are supported via "language environments" that translate between the source language and *Lingua Franca*. Note that not all environments allow translation in both directions; some language environments only translate from the source notation to *Lingua Franca* (and may only be used to create script), whereas other only translate from *Lingua Franca* to some other notation (and may only be used to display script). The most general class of language environments perform translation in both directions; these may be used to edit a script, by first translating from *Lingua Franca* to a "source" notation, modifying that representation, then finally translating it back to *Lingua Franca*. To allow this bidirectional transformation, language environments must conserve all information in the *Lingua Franca* representation, regardless of whether it is meaningful in the present language or not. (Contrast this to traditional compilers, where information and structure not relevant to the

result is usually discarded.)

The two types of information that are most commonly discarded when translating a script from one form to another are *secondary notation*, such as comments, and *higher level structure*, such as loops. Both of these may vary greatly from language to language. *Lingua Franca* allows multiple secondary notation elements to be associated with a part of a script; each such element is tagged with a notation type, to allow language environments to determine which (if any) to display. Higher level structure is represented by grouping; again, each group is tagged with a type (such as "while loop"), which may imply a particular structure, and language environments may use this to determine how to display the group's members. Unlike secondary notation, any environment that can display *Lingua Franca* can display any group, as in the worst case it can simply display it as a grouped collection of primitive operations.

We have implemented a interpreter that stores the "corpus" of scripts that have been entered into the system. Language environments communicate with this interpreter via HTTP, allowing them to read, add to and update the *Lingua Franca* code (represented as XML) that makes up the corpus. In addition, the interpreter is responsible for executing *Lingua Franca* code, and interfacing the *Lingua Franca* environment with the rest of the ubiquitous computing system.

A MENAGERIE OF PROGRAMMING LANGUAGES

A wide variety of scripting languages are being developed in order to demonstrate the flexibility and range of the *Lingua Franca* architecture. These languages are designed to complement each other, in that they may be used to perform different manipulations on the same script with ease. Each language is embodied in a *language environment* that provides an interface via which the user can view and/or manipulate a particular notation, translates between the notation and *Lingua Franca*, and communicates with the *Lingua Franca* interpreter via HTTP.

A *textual language* provides an interface familiar to those with experience of conventional scripting languages. It is envisioned that this will be primarily used for editing substantial scripts, a task most likely to be undertaken by someone with at least some programming background. (It is of course possible to manipulate *Lingua Franca* directly in XML form, but this is needlessly difficult and carries the risk of introducing malformed code into the database, or accidentally removing or modifying data associated with another language environment.)

Two forms of *visual language* are in development, serving slightly different needs. The first is a purely *presentational* diagram that cannot be used to create or edit scripts, but only to display them. This allows it to be specialized in order to facilitate searching, navigation and comprehension of scripts. The second, a *mutable* diagram, allows scripts to be edited, and is likely to be the main environment for the manipulation of mid-sized scripts.

Perhaps the most unusual of the language environments being developed for use with *Lingua Franca* is the *Media Cubes* language. This is a "tactile" programming language, in other words, a language where programs are constructed by manipulating physical objects—in this case, cubes augmented such that they can determine when they are close to one another. The faces of the cube signify a variety of concepts, and the user creates a script by placing appropriate faces together; for example, to construct a simple radio alarm clock, the "Do" face of a cube representing a conditional expression would be placed against a representation of the act of switching on a radio, and the "When" face against a representation of the desired time. In an appropriately instrumented house, the representation can often be an existing, familiar item, or even the object itself. In the above example, a time could be represented using an instrumented clock face, and turning the radio on could be represented by the radio or its on switch.

The *Media Cubes* language is intended to be easy for those unfamiliar to programming, and as such would provide a low-impact path from direct manipulation to programming. However, the language as it stands is unusual in one very significant respect—scripts do not have any external representation. This means that it is only feasible to construct small scripts, and that, once created, scripts may not be viewed, and hence may not be modified. However, as the language exists within the *Lingua Franca* framework, we do not need to abandon the language, with its substantial advantages. *Lingua Franca* makes it feasible to include niche languages such as the Media Cubes in a system without sacrificing functionality.

REFERENCES

1. Blackwell, A.F., Hewson, R.L. and Green, T.R.G. (2003) Product design to support user abstractions. *Handbook of Cognitive Task Design*, E. Hollnagel (Ed.), Lawrence Erlbaum Associates.

2. Blackwell, A.F. and Hague, R. (2001). AutoHAN: An Architecture for Programming the Home. *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, pp. 150-157.

3. Blackwell, A.F., Robinson, P., Roast, C, and Green, T.R.G. (2002). Cognitive models of programming-like activity. *Proceedings of CHI'02*, 910-911.

4. Green, T.R.G, Petre, M. and Bellamy, R.K.E, Comprehensibility of visual and textual programs: A test of superlativism against the 'match-mismatch' conjecture, *Empirical Studies of Programmers: Fourth Workshop*, J. Koenemann-Belliveau, T.G. Moher, S.P. Robertson (Eds): Norwood, NJ: Ablex, 1991

5. Peyton Jones, S., Blackwell, A and Burnett, M. (in press) A user-centred approach to functions in Excel. To appear in proceedings *International Conference on Functional Programming*.

Towards Ubiquitous End-User Programming

Rob Hague, Peter Robinson & Alan Blackwell

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

We believe that end-user programming capabilities are an essential part of any flexible ubiquitous computing system, allowing users to add functionality that was not, and in many cases, could not have been, anticipated by the system's designers, and thus enabling them to benefit fully from the possibilities ubiquitous computing offers. The *Lingua Franca* system is designed to allow end-user programming in multiple notations, in order to adapt to the large variation in users, tasks and communication channels present in a ubiquitous computing environment (specifically, the networked home).

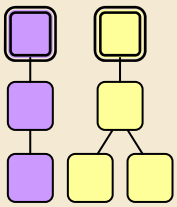
Lingua Franca is an XML-based intermediate representation of scripts that permits translation both to and from source notations. Scripts are stored in a central database with which *language environment* communicate over the network.

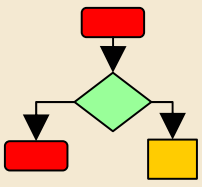
Lingua Franca

```
<group name="Shops">
  <receive>
    <nt>GoShopping</nt>
    <dispatch>
      <nt>Order</nt>
      <nt>Milk</nt>
    </dispatch>
  </receive>
</group>
```

- Allows a single program to be manipulated via multiple notations.
- Transformations preserve structure and secondary notation.
- Facilitates automatic processing, e.g. for policy enforcement.

```
[Shops]
?GoShopping ->
  {!Order/Milk}
[/Shops]
```





A traditional *textual* language will be provided for use by those familiar with conventional programming languages. This is likely to be used by "power users" for complex scripting tasks. (It is possible to edit *Lingua Franca* XML directly, but doing so is not advisable as it bypasses integrity and policy checks.)

A *visual language* will provide a means for users with a variety of levels of experience to both view and edit scripts in a straightforward manner.

A second visual language will be provided for *presentation*; this notation does not need to support editing, and hence can be optimised for navigation and searching.

The *Media Cubes* are a "tactile" language in which programs are constructed by manipulating physical objects. The notation can only be used to create programs, not to view or edit them.

Lingua Franca allows other notations to be used for these tasks, making specialised languages such as this feasible.

RAINBOW RESEARCH GROUP

William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
http://www.cl.cam.ac.uk/