McBryan, Tony (2011) *A generic approach to the evolution of interaction in ubiquitous systems.* PhD thesis.

http://theses.gla.ac.uk/2620/

A Generic Approach to the Evolution of Interaction in Ubiquitous Systems

by

Tony McBryan

Abstract

This dissertation addresses the challenge of the configuration of modern (ubiquitous, context-sensitive, mobile et al.) interactive systems where it is difficult or impossible to predict (i) the resources available for evolution, (ii) the criteria for judging the success of the evolution, and (iii) the degree to which human judgements must be involved in the evaluation process used to determine the configuration.

In this thesis a conceptual model of interactive system configuration over time (known as *interaction evolution*) is presented which relies upon the follow steps; (i) identification of opportunities for change in a system, (ii) reflection on the available configuration alternatives, (iii) decision-making and (iv) implementation, and finally iteration of the process.

This conceptual model underpins the development of a dynamic evolution environment based on a notion of *configuration evaluation functions* (hereafter referred to as *evaluation functions*) that provides greater flexibility than current solutions and, when supported by appropriate tools, can provide a richer set of evaluation techniques and features that are difficult or impossible to implement in current systems. Specifically this approach has support for changes to the approach, style or mode of use used for configuration - these features may result in more effective systems, less effort involved to configure them and a greater degree of control may be offered to the user.

The contributions of this work include; (i) establishing the the need for configuration evolution through a literature review and a motivating case study experiment, (ii) development of a conceptual process model supporting interaction evolution, (iii) development of a model based on the notion of evaluation functions which is shown to support a wide range of interaction configuration approaches, (iv) a characterisation of the configuration evaluation space, followed by (v) an implementation of these ideas used in (vi) a series of longitudinal technology probes and investigations into the approaches.

Acknowledgements

Contributory Papers

[1] Philip Gray, Tony McBryan, Chris Martin, Nubia Gil, Maria Wolters, Neil Mayo, Ken Turner, Liam Docherty, Feng Wang, and Mario Kolberg. A Scalable Home Care System Infrastructure Supporting Domiciliary Care. Technical Report CSM-173, Department of Computing Science and Mathematics, University of Stirling, UK, 2007.

[2] Tony McBryan and Phil Gray. A Generic Approach to the Evolution of Interaction in Ubiquitous and Context-Aware Systems. Technical Report TR-2007-260, Department of Computing Science, University of Glasgow, 2007.

[3] Tony McBryan and Phil Gray. A Model-Based Approach to Supporting Configuration in Ubiquitous Systems. In *Design, Specification and Verification of Interactive Systems 2008*, Kingston, Ontario, Canada, 2008.

[4] Tony McBryan and Phil Gray. A Framework for Runtime Evaluation, Selection and Creation of Interaction Objects (Poster) . In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, CMU, Pittsburgh, PA, USA, 2009.

[5] Tony McBryan and Phil Gray. User Configuration of Activity Awareness. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Salamanca, Spain, 2009.

[6] Tony McBryan and Phil Gray. Using Activity Awareness as a Run-time Interaction Configuration Testbed (Poster). In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, CMU, Pittsburgh, PA, USA, 2009.

[7] Tony McBryan, Marilyn R McGee-Lennon, and Phil Gray. An Integrated Approach to Supporting Interaction Evolution in Home Care Systems. In *1st International Conference on Pervasive Technologies Related to Assistive Environments (PETRA)*, Athens, Greece, 2008.

[8] Marilyn McGee-Lennon, Maria Wolters, and Tony McBryan. Audio Reminders in the Home Environment. In *Proceedings of the International Conference on Auditory Display (ICAD)*, Montreal, Canada, 2007.

Table of Contents

Table of Figures

1

Introduction

Ubiquitous Computing (ubicomp) [230], also described as pervasive computing [196], proposes a grand vision of complete integration of computing facilities into everyday life. Within such an environment a user might interact with hundreds, if not thousands, of small inexpensive sensors and feedback devices throughout the course of their day.

Ubiquitous computing is, by its very nature, embedded in a world of change [5]. The people in the environment will change their current tasks, they will move location and their preferences will evolve over time; therefore a ubiquitous environment must be able to be configured, and reconfigured, to support simultaneous tasks that are currently under-way, to accommodate change of context or function for those tasks and to support addition of tasks which have not yet been started or even conceived of.

This requires a system architecture that is able to adapt to accommodate devices developed after the system has been installed. The recognition of constant change of the contextual environment of a ubiquitous system results in a further requirement for systems which can evolve and adapt to these changing circumstances [237]. There are a number of different sources of change (for example environmental or temporal) which will be discussed in Chapter 6.

However, the problem of change needs to be further addressed. It is not always clear how the system would be expected to change. What rules or policies are responsible for deciding

how the system changes? Should all change be manually driven - requiring almost constant intervention with the user to decide what to do next or should change be automatic with little or no control over how change occurs?

Controlling and directing change is not an easy task [20]. It is not possible for the designer of such a system to decide how to reason about change in a way that pleases everyone [132]; therefore ubiquitous systems need to be able to flexibly adapt to change.

The choice of interactions to use and the different benefits and drawbacks of different interaction-modalities and user interface widgets is important to the successful design of usable systems [130]. Likewise, we encounter this problem when designing physical products - it must be decided how the product will interact with the user and vice-versa.

In many cases these problems do not only apply to a single person but also to friends, family and other stakeholders [148]. The method of interaction with ubiquitous systems would be forced to change according to the context of the world they operate in; which includes the fact that other people will be living and working in the same spaces. It is therefore necessary to be able to adapt, not only to one person, but to those living and working in the same spaces.

This is complicated by the arrival of new devices or techniques aimed at addressing an existing or new problem, but which do not interoperate with existing solutions.

Models exist to help a programmer or designer to reason about these interactions and will be discussed in Chapter 2; however, systems are emerging that attempt to support multiple types of interaction within the one system where it is possible to create interactions and combinations of devices and tasks at runtime instead of at design time. An artefact of this is that the interactions would not be designed by the creator of the system, but instead would be designed by the end user.

This thesis investigates approaches to supporting the process of change to allow for configuration to take place within a ubiquitous system. Particular emphasis is placed on allowing this process to be undertaken by the users of such a system at runtime rather than the developers to create a flexible approach to configuration.

1.1   Research Questions Motivating this Work

Given the requirement for change and evolution established in the previous section, this work investigates how such evolution might be supported. When this research began, it was motivated by a research question related to the provision of support to the users

of ubiquitous systems, viz. how might a system enable a user to answer the following questions:

- What is the system currently doing?

- What can it do?

- How can it be changed?

However, it soon became apparent that an answer to this question would depend on how that system modelled (potential) change and how such a model or models were incorporated into the runtime system. For that reason, the work has focussed primarily on a derived research question:

- *How can system change be modelled and implemented in order that the system can enable a user to answer the questions above?*

1.2 Thesis Statement

This thesis proposes an approach and supporting model where specific configuration criteria are delegated to a structure known as *configuration evaluation functions*, (hereafter referred to as evaluation functions), which are responsible for a portion of the configuration decision making process. The evaluation functions operate by inspecting, filtering, ranking, sorting and manipulating the available configuration options which can be adopted and these functions can be themselves manipulated and combined with each other in order to provide a flexible tool for system configuration.

Therefore it is claimed that *a model-based approach to the dynamic configuration of interactive systems, based on the concept of generic and specialisable configuration evaluation functions provides benefits in terms of design, implementation and use of interactive systems and is a suitable approach to modelling interaction evolution in order to answer the questions posed in the previous section.* Specific subclaims of this thesis are that:

- As these evaluation functions can support varying modes of use and can be combined to allow novel support for the configuration of interactive systems.

- Such an approach allows for systems that encompass currently available techniques used for the configuration of interactive systems and allows for them to be used and combined coherently within a single system.

3

- The approach provides greater flexibility in terms of configuration techniques than is currently possible, such as the capability to change mode of use or criteria for configuration choice at runtime, which is unavailable or difficult to implement in currently existing approaches.

- The approach provides users with information on system capabilities and status which can be difficult to determine otherwise and allows them to make more informed decisions to alter the behaviour of the system.

With respect to the questions proposed in Section 1.1 this dissertation proposes a solution that addresses the key questions as follows.

What is the system currently doing?: The proposed mechanism for evolution in this thesis explicitly models configuration options as *possibilities* which can be enumerated and inspected and upon which evaluation functions operate. Inspection of the possibilities which have previously been selected for implementation allows the user to determine the current configuration.

What can it do?: This thesis proposes a novel approach for the determination of the currently available set of configuration options (modelled as possibilities) via a graph traversal operation; thereby allowing an exhaustive set of possible alternate configurations to be determined. Evaluation functions can be used to inspect, filter, rank, sort and manipulate the set of available possibilities.

How can it be changed?: The mechanism for changing the current configuration is undertaken by control and manipulation of the evaluation functions which are responsible for selecting which possibilities are actually implemented within the application. Users select evaluation functions which accurately model the user (or users) requirements and preferences in order to change how the system behaves.

This approach offers the ability to explicitly model decision events which are responsible for change and evolution within a ubiquitous system. The model allows incorporation of a wide range of techniques and approaches to configuration choice - including automated decision making, user driven choices, context sensitive adaptation and criteria from multiple stakeholders - and allows them to be used jointly and concurrently within the same system. A full discussion of this configuration space and how it can be supported within this model is presented in Chapter 6.

1.3 Research Scope

This thesis is intended to apply to interactive systems in general but particularly ubiquitous computing applications. As such this thesis focuses on systems which exhibit features that are characteristic of those in ubiquitous computing: component-based, distributed, context aware, multimodal and/or mobile.

This work was undertaken within the context of the MATCH (Mobilising Advanced Technology for Care at Home) project[1]; a seven year collaborative research project between the Universities of Dundee, Edinburgh, Glasgow and Stirling, supported by the Scottish Funding Council. The project is focused on developing technologies for care of elderly or vulnerable people at home and is intended to benefit people with infirmities or disabilities who would prefer to stay at home rather than move to expensive clinical or social care facilities. MATCH is involved in developing home care technology in four key areas:

- home networks: provision of care services, flexible service discovery, and policy-based management of home care

- lifestyle monitoring: automated recording and analysis of daily behaviour, identification of trends, and deterioration in the users condition

- multi-modal interfaces: use of speech and non-speech audio, haptic interfaces

- evolutionary configuration management: ability to adapt to context, mode of use and changing requirements

The work done during the course of this research within the MATCH project incorporates elements of each of the four themes identified above but only the discussion of configuration is within the scope of this thesis.

Due to an ageing population [222] increasing numbers of people will require care in the future and it is economically and socially preferable to offer these people care within the comfort of their home rather than requiring them to move into specialised care facilities. Ubiquitous systems (and Smart Homes in particular) are hoped to be able to assist elderly people, and others suffering acute and chronic conditions requiring care, by providing alerting, monitoring and control abilities to the user which would otherwise need to be done by paid support or nursing.

There are particular challenges in the context of homes designed for Telecare or in the homes of elderly or vulnerable people which have been fitted with technology to support

---

[1]http://www.match-project.org.uk/

care as these additions can add an additional level of complexity via the nature of the applications, users, physical settings etc. The situation of someone with a long-term illness is subject to great volatility as their conditions may get better, or worse over time. Situations will change continually within the home; requiring changes in any supporting infrastructure designed to assist the person. These users are likely to have extremely different requirements on a daily basis as well as long term changes as their condition changes and as such the systems and services they use to support themselves at home must be capable of adapting to changing circumstances in both short and long terms.

As this work was undertaken as part of a project related to homecare it, consequently, has had to address a broader set of issues that the project was tackling and also to use and develop software in collaboration with other project members. As such, usage of specific speech synthesis tools, challenges of integrating tools and models with policy and resource management and collaboration with older participants linked to the project are all themes which arise from this participation and are visited in this thesis.

Due to the relevant and challenging situations relating to ubiquitous computing a particular emphasis has been placed on the requirements of this category of system and on developing prototypes in that area. Nevertheless, it will be argued that the key claims are applicable to the broader class of systems and contexts of use.
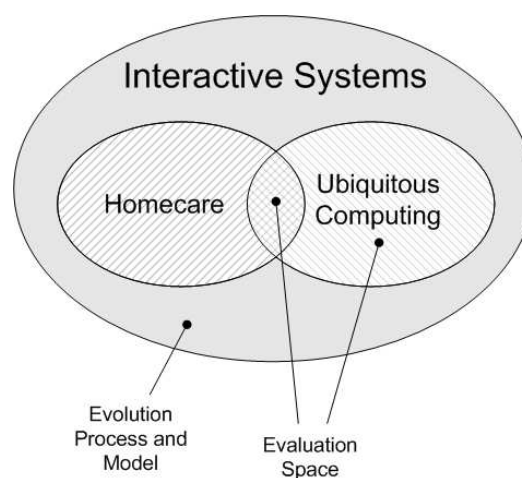
Figure 1.1: *Research Scope illustrating relevant areas in which this thesis will take place*

Figure 1.1 shows a diagrammatic representation of the intended scope of this work. The concepts behind this research are grounded and applicable in the area of Interactive Systems modelling while the examples and applications built during this research are implemented within the ubicomp domain but with several applications or examples drawn from the

homecare area due to the explained context of the MATCH project.

1.4   Research Approach

This thesis involved a number of related work items which combine to address the research question. In particular, the approach of this research included:

- an investigation into the need for configuration

- a process for change within which evolution is supported

- a model supporting evolution of configuration

- an exploration of the model's configuration space

- an implementation of the model for exploration comprising a justification in terms of:

    - feasibility of the model

    - scalability of the model

    - flexibility of the model

- user interaction studies

Each elements of the approach is an important contribution to this thesis. The initial investigation addresses the need to build solutions to this problem and shows why the techniques within this thesis are necessary.

The conceptual process is needed to understand the high level mechanics of change that take place during configuration. A model is developed which operates within this process and provides for the ability to compare and contrast configuration possibilities in order to determine how the configuration should proceed.

The range of configuration approaches that can be used within the evaluation model is explored - describing how they can be implemented within the model and discussing their relative advantages and disadvantages.

A Java based implementation of the model is presented. In order for this model to be of any use it must be shown that it is feasible (it works), it is scalable (it works on more than just toy examples) and that it is flexible (it can be used in a range of situations). This is shown by demonstration applications, performance and scalability measurements and by application of the implementation in a variety of situations.

User studies are intended to show the proposed user benefits (usability) of the model and its impact on the system evolution process.

This approach involved model development, interactive systems patterns, architectural investigation and user- based studies based on a prototype. It is believed that the steps taken in this approach each contribute to answering the questions originally posed in Section 1.1.

1.5   Overview

The remainder of this report is structured as follows:

Chapter 2 explains the notion of configuration and evolution including details of the types of interactive system to which evolution can be applied as well as previous related work.

Chapter 3 presents a motivating empirical experiment which demonstrates the need for evolution within ubicomp applications as well as highlighting some of the issues arising from the necessity of making choices concerning configuration.

Chapter 4 details the process of evolution that influences the model proposed in this thesis. The model itself is discussed in Chapter 5 and further explored in Chapter 6 along with a characterisation of the configuration space and a discussion of how a disparate range of evaluation approaches can be integrated within the model.

An implementation of the model is discussed in Section 7 which has been used in a variety of different projects to demonstrate its flexibility. Two longitudinal user investigations, using applications built using the framework, are discussed in Section 8 and explore the processes used by users during configuration tasks.

Chapter 9 lays out foundations and directions for future work, which can build upon the ideas presented here, while Chapter 10 concludes.

Figure 1.2 provides a visual map of the thesis structure and shows how each chapter within this thesis influences the successive chapters. This map demonstrates the dependencies between each chapter in the thesis and shows a number of possible reading approaches.
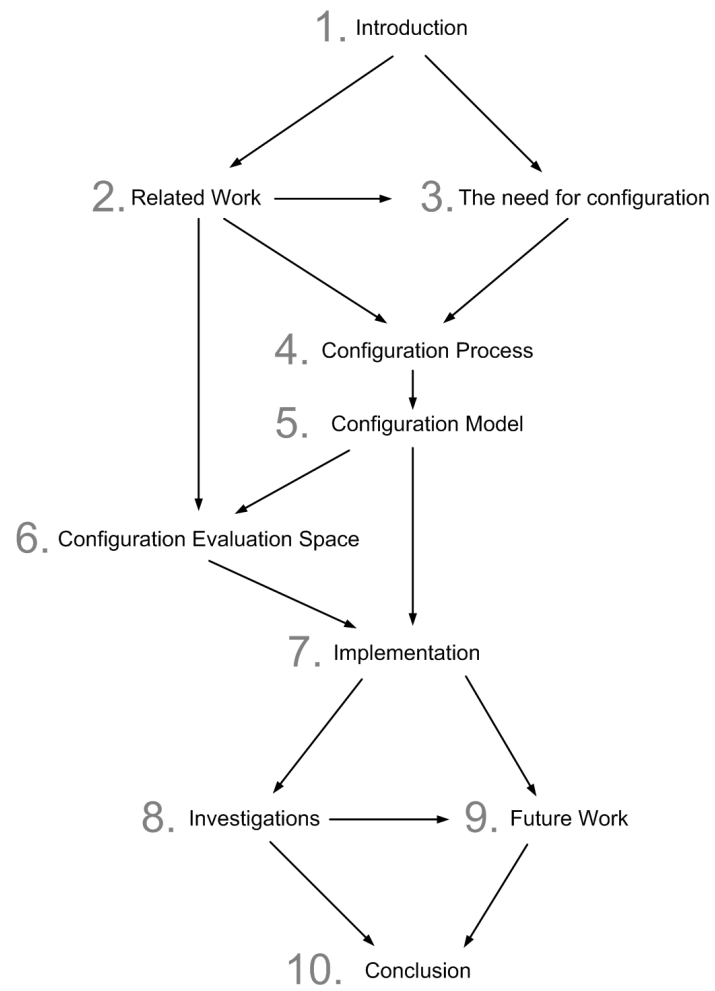
Figure 1.2: *Thesis Map*

2

Related Work

This chapter presents a summary of existing work in interaction configuration theory, approaches, methodologies and applications. This summary provides an overview of the current state of the art within the field and explores a variety of different configuration ideas and concepts with the view of applying them to interactive systems.

2.1 Types of Configuration

This chapter will start by discussing configuration in the general sense and detail the different types of configuration that are common in current interactive systems. Discussed here are configuration techniques centred on customisation, mass customisation, personalisation and evolution.

The following definitions are made in this discussion of configuration:

- **Component** - a functional unit within a system;

- **Configuration** - a collection of functional units, which may be connected, which is complete enough to fulfil some or all of the goals of the system;

- **Configuring** - the selection of components, services or features to better suit the users needs or the requirements of the application;

- **Reconfiguring** - changing the selection configuration (repeated configuring);

- **Evolution** - multiple related directed instances of reconfiguration.

Additionally, in this discussion of configuration the following terms are used while introducing the history of configuration.

- **Customisation** - supplier driven configuration of a product within a fixed set of options;

- **Mass Customisation** - customisation on a large scale;

- **Personalisation** - user driven customisation where the user provides their own configuration options;

- **Adaptive System** - a system which is capable of changing its behaviour in response to an internal or external change.

The following sections elaborate on the relationships between these definitions.

2.1.1 Customisation

It has long been possible to get a product in a customised form; different from the product offered to the general public. This customised version may cost more (via price discrimination [221]), be hand-made rather than machine produced [44] and have been used as status symbols [227]. Tailoring a product directly to a specific customer can be more expensive than producing the same one but it makes it more likely that the customer would purchase your version of the product rather than your competitors. For example, National Bicycle require triple the labour to produce custom products when compared to assembly-line production but affluent customers are willing to pay more for the resulting product [78] making the tactic a successful one.

Solomon et al. [212] claim that the availability of customisation is a basic factor that customers consider when evaluating products which indicates the importance of this factor in the purchasing decision. Gardyn [86] has shown that the customer is willing to pay more for a customized product. Both Solomon and Gardyn reinforce the simple fact that users want to be able to customise products and services to better cater for their own requirements.

The term customisation is used to refer, primarily, to the single customisation of a product for a particular user or customer. As a software parallel, consider the development of variants for a single customer, or class of customers, as customisation. For example,

branded Internet Explorer editions [1] used by internet service provider's (ISP) constitute a particular customisation option offered by Microsoft to the ISP. It is natural to consider "editions" of software to be very coarse customisation options (as well as instances of price differentiation) provided by the supplier of the software - a typical example would be the multiple editions of Windows Vista [157] targeted at different classes of users.

2.1.2 Mass Customisation

The concept of Mass Customisation was introduced by Davis [50] who said that it was the ability to provide individually designed products and services to all customers.

According to Kotha [121] the ability to utilise Mass Customisation can offer a significant competitive advantage in the market place due in part to customer preference for customised products as well as more favourable public image of the company as being innovative and customer-driven. The advantages of Mass Customisation in the market place are demonstrated by the success of Dell Inc. becoming one of the largest supplier of personal computers [38, 211] using a direct sale model offering customers the ability to customise a PC for their specific requirements [54].

Da Silveria et al. [208] discuss the enabling techniques and technologies for Mass Customisation, among these is the fact that "Knowledge must be shared" - this is in reference to the product designer needing to be aware of the customers demands before they can offer those customisation options. The product itself must be capable of being customised with those options but the product must be built with these options in mind. However, it is generally not possible to predict with certainty what customisation options are desired - which means that early adopters may not be offered the customisation options they want.

Hart [103] shows that there are effective practical limits to the range that a product can be meaningfully differentiated for each potential customer using Mass Customisation, effectively forming an "envelope of variety" caused by technical, managerial and logistical limitations involved in providing large numbers of customisation options.

An example of Mass Customisation within the software industry is the provision of configuration files supporting a fixed number of options which can be altered by the user. A concrete example of this would be the SSHD (Secure SHell Daemon) configuration file provided with OpenSSH [15] which, for example, has a *PermitEmptyPasswords* field which

---

[1]Yahoo! Inc. Internet Explorer 8 optimized for Yahoo! - http://downloads.yahoo.com/internetexplorer/

can be either "yes" or "no". That this option is available for anyone using the software to customise makes this an instance of Mass Customisation.

2.1.3 Personalisation

Personalisation is an extension to the idea of customisation where the tailoring of the product is based on the individual users attributes, desires and preferences rather than on the characteristics of a group of customers. This means that the customisation options are personalised instead of being the same options offered to every user.

As an example, consider the SSHD configuration file option presented in the previous section. Imagine a particular person wanted to allow only particular user accounts to have empty passwords but the configuration file would not support this. The feature could be added to the configuration file; but perhaps an additional feature that would be desired would be to allow only particular user groups to have empty passwords, or perhaps only allow empty passwords within a specific time period of the day. There are potentially limitless sets of configuration options that could be added and it would be clearly impractical to include every possible configuration - especially since every new option that can be added is unneeded by most users. Since a user needs to understand the option in order to decide if they need it it may make it more difficult for people to use the configuration file if the number of options increases by orders of magnitude.

Allen [2] defines personalisation as an interactive conversation between the supplier and the user, effectively creating an envelope of variety for each customer based on their specific customisation requirements. This conversation is used to define the personalisation options and implement them. Rather than being forced to choose from a set of pre-configured options the user can introduce their own options into the personalisation process.

A key difference between customisation and personalisation is the driver behind the configuration. In customisation the provider of the product or service recognises the need to configuration to be adjustable for individual users and offers the user that ability resulting in a process driven by the provider. In personalisation the user who recognises their own need for configuration changes drives the process; personalisation may be supported by the provider but is not driven by them.

It should be clear that although the user drives personalisation it must be supported by the supplier of the software. An example of personalisation in wide scale use is the iGoogle customised homepage [27] which can have "gadgets" [95], created both by Google and 3rd party developers, added to the page. Here the user recognises they want a new feature on

their homepage and either seek out a gadget to fulfil that function or create one themselves. Google has therefore provided the personalisation space of adding gadgets to the page but it does not limit the choice of gadgets to only those provided by Google.

Customisation and personalisation do not exist independently of each other and a single configuration option may be subject to both customisation and personalisation simultaneously. For example, an SSHD configuration file may provide a choice of two host key algorithms to use as well as allowing the choice of the actual host key to use. The choice of algorithm would be regarded as customisation of algorithm because it is a choice of fixed options provided by the designer of the software, but since the host key itself is provided by the user this aspect would be regarded as personalisation of the host key itself. Similarly organising the order of items in a menu would be customisation, while renaming the items or adding your own would be personalisation.

2.1.4 Evolution

Personalisation of a product or service primarily refers to a specific instance of personalisation where the user identifies a missing or substandard component and adds or replaces it with something new. Instances of personalisation may not be independent, although they may be treated as though they are by the provider. Evolution of a system can be regarded as a natural extension of this where personalisation is ongoing which results in a system that evolves through sequences of personalisations. These instances of personalisation which underpin evolution may be independent or may be related and goal driven.

It is this persistent idea of configuration evolution that provides support for MacLeans "tailoring culture" [133] where continual reconfiguration of a service or product is regarded as the norm. MacLean refers to the ability of small scale incremental improvements being able to diffuse throughout a user community as the principle behind user driven evolution.

Dourish [57] states that a developer must be concerned not only with the traditional software design issues but must ensure that any systems they design are amenable to evolution by the user. Dourish goes on to state that this evolution is important to allow the user to adapt the software to their particular needs.

Fickas introduced the idea of clinical requirements engineering [74] as a method of requirements gathering, combined with clinical methods, which exhibits aspects of the evolution of configuration and reinforces the need for systems that can evolve to meet the users requirements. Fickas uses a "goal attainment scale" to represent the change in ability of a cognitive rehabilitation patient; however this is done at design time rather than the more

flexible approach that evolution offers.

Evolution is most powerful when the goals are not explicitly known in advance or, as with Fickas, circumstances are changing throughout the lifecycle of the service. Evolution can be used to experimentally test different configurations by the user to determine if they are suitable, and if a configuration is unsuitable or becomes unsuitable then it can be changed. The goals in evolution may be unspecified or specified only in respect of global goals rather than aims for specific interactions. These goals may be considered as a set of functions which guide the evolutionary process.

The need for constant reconfiguration of devices within a home environment has been discussed by O'Brien and Rodden [165, 190] who recognise that the home is subject to continuous redesign. This evolution within the home environment has a knock-on effect on interactive systems operating in that space - requiring them to be capable of evolving to deal with the new or changing requirements. O'Brien and Rodden present a simple real world example they obtained during ethomethodological studies where even the physical arrangement of furniture is subject to reconfiguration based on the expectation of guests - the users in this study would reconfigure their furniture to focus on more "social" devices such as music players rather than the usual arrangement that was television-centric.

2.1.5   Adaptive Systems

A tempting phrase to use when discussing systems that are configurable in ways similar to evolution is to describe them as *adaptive systems*. Benyon [21] describes adaptive systems as a system which has a mechanism to automatically select alternative behaviours.

There is a distinction drawn between the concept of adaptation and evolution by reinforcing the notion that an adaptive system describes a system which is capable of adapting on its own but that that the term evolution describes a system that, although it may adapt itself, can be adapted by external agents directly modifying the goals of the system. Evolution has the notion of an inherent (although possibly unspecified) goal or evolutionary pressure guiding changes and selecting against poor configurations - something that is a strong influence in the work in the next chapter.

2.1.6   Social Aspects of Configuration

Although the social aspects of configuration are not the direct focus of this work they still warrant discussion as the act of customisation is recognised as both explicitly and implicitly

social activity.

Explicit social activity with respect to a customisation occurs primarily when the configuration affects multiple people who use, or rely on, the system who may have multiple, possibly conflicting, demands on the system. McGee-Lennon and Gray [150] list some of the activities involved here as the identification, negotiation and resolution of conflicts between different stakeholders. McGee-Lennon and Gray refer to this stakeholder conflict in the context of home care systems but it is applicable in a broader sense including ubiquitous systems and computer supported collaborative work in general.

Mackay [131] states that although customisation of software is often viewed as a solitary activity this is not the case. Mackay provides two real life examples of configuration sharing even with standalone software. Mackay identifies particular roles within the community such as "rule gurus" who produced large numbers of useful configurations and "translators" who were responsible for much of the communication regarding the rules. Mackay offers the example of a particular subset of a configuration for filtering uninteresting email (known as the "boring rule"). The rule was able to propagate within the user community but that attempts to copy the entire configuration of another user verbatim were prone to failure - emphasising that the configuration as a whole often only makes sense for a particular user but that individual parts of the configuration are more amenable to sharing.

Where configurations are shareable socially the "survival" (its prevalence in the user community) of a particular configuration is a direct result of its usefulness. This can be compared with Darwinian evolution [49] where survival of the species is linked to reproductive success. For a configuration option to survive it must, effectively, compete with alternative options and/or co-exist with complementary components as with symbiotic relationships [137] between different animal species. The previous example of the "boring rule" in Mackay's work illustrates this; the configuration was able to survive and propagate to other users due to its usefulness to the users but other configurations, that were not as useful to that particular user, were abandoned.

2.2   Configuration Targets

The previous section discussed the different types of configuration that can be employed as well as discussing their relationships. This section will expand on this by discussing the types of things that users are likely to want to configure using these techniques. This discussion has particular relevance to the work presented in this report by grounding later work in concrete and, reasonably understood, interactive systems designs, implementations

16

and techniques.

2.2.1 Ubiquitous Computing

Originally coined by Weiser [230] Ubiquitous Computing (often abbreviated to UbiComp) is the idea that, eventually, technology will be tightly integrated into everyday objects and activities. Weiser originally proposed *tabs*, *pads* and *boards* as 3 particular exemplars of physical pieces of technology, existing in different scales, which he predicted would be in common use in such a ubiquitous environment. Although there are no fully integrated systems, yet, we have come close to practical, commercial implementations of several of his ideas [75, 89, 114, 174]. A ubiquitous computing system would be virtually constantly interacting with the users of such as system due to the prevalence of the system - it will have direct impacts on daily life.

The ubiquitous ideals hope for the complete integration of technology into the physical world but it is not the case that this integration must be always seamless [35] and transparent. On the contrary it is proposed that design of ubiquitous systems must recognise the presence of seams - one of these seams would be configuration techniques.

There are a huge number of configuration elements in a ubiquitous computing world - ranging from simple preference settings for ambient light levels to expressing complicated interdependent policies to govern the behaviour of the environment - but this work will concentrate on the configuration options most relevant to controlling and configuring interactions within a ubiquitous computing system.

2.2.2 Context Aware Systems

Context aware systems originated from the work done on ubiquitous computing and the term was first used by Schilit and Theimer [199] where they describe a context aware system as one that can adapt according to location, nearby people and objects as well as changes to location and objects over time.

Location is often cited as the primary factor in context aware systems but Schmidt [200] points out that this is far from the only factor. Schmidt provides a hierarchical model of context which places the user model, social environment, task model, environmental conditions and physical infrastructure on the same "level" as location. These categories can be further classified; for example the environmental conditions may include light, pressure, audio and temperature. These classifications can be further divided to extract particular

features. The features of light might be the level of light, flickering, wave-length (natural light or artificial) and so on. It may not be possible to have sensors to actually measure all of these factors but this may not always be the case.

A simple example of a social factor that may be applicable in context aware systems is obtained from user studies by McGee-Lennon et al. [152] where users expressed particular preferences for the form of an audio reminder in the home depending on if they were currently alone or had guests.

The aspect of context sensitive computing most relevant to this work is that the configuration of a context sensitive system cannot be static and unchanging while everything about it changes - it needs to be capable of evolving configurations as the context it lives in itself evolves. Therefore context sensitive systems are an ideal test-bed for experimentation and demonstration of evolving configuration techniques.

2.2.3 Home Care Technologies

One particularly suitable target which encapsulates both ubiquitous computing and the need for context sensitivity is that of home or office automation. Here the focus is specifically on home automation due to the influence of the MATCH project on this thesis. Home care or home automation solutions are usually designed to mitigate risks that an elderly person on their own might encounter or to enable remote or automatic usage of devices (lights etc).

The first form of homecare system is usually aimed as telecare (or telehealthcare) solutions. A typical supplier of this type of solution is Tunstall [220] who provide a number of prepackaged solutions aimed either at individual homes or at grouped or social housing facilities. The hardware provided can include a variety of proprietary sensors such as bed occupancy detectors, fall detectors, movement detectors, alarm buttons, environmental monitoring (gas, fire, intruder detectors) and will raise a warning to a staffed monitoring centre should an undesired event occur. Although the selection of sensors can be personalised (to some degree) for a particular resident and the thresholds for dangerous conditions adjusted; the ability to configure these systems is very limited.

At the opposite end of the spectrum are approaches which allow static user configuration based on conditions and actions [18]. One example is Indigo [170] which allows control of X10 based devices (light switches, power sockets, remote controls, motion detectors) based on a collection of rules or triggers, shown in Figure 2.1. This approach allows for a static configuration and is particularly fragile in respect to configuration changes.

Another approach taken within home automation is to create connections between physical
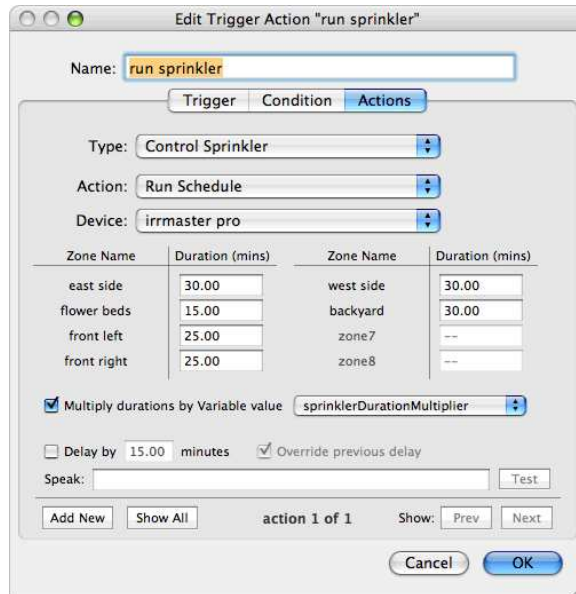
Figure 2.1: *Indigo Configuration*

or logical devices as taken by the Jigsaw [109] and OSCAR [161] approaches. Figure 2.2 is from the OSCAR project which designed to allow users to connect media devices; either logical devices, representing music libraries, or physical devices, representing speakers or cameras, together through the home through a process of composition where one device represents the sink while another is the source. Although these connections can often be temporarily disabled to allow another connection to be used - it is not generally possible to allow these connections to be context sensitive.

Each of these configuration approaches are discussed in more detail later in this chapter.

2.2.4   Component Systems

This work could be implemented without relying on the idea of component based development [205] but they do provide a useful architecture to experiment with the techniques proposed later in this thesis. A component based system allows addition and removal of components which can be used as an additional form of change that the evolution of the configuration must be able to adapt to. Component based systems therefore provide a simple mechanism for applying changes of context to a running system by changing the available service components that are available.

A software component can be described as a single unit of software intended for
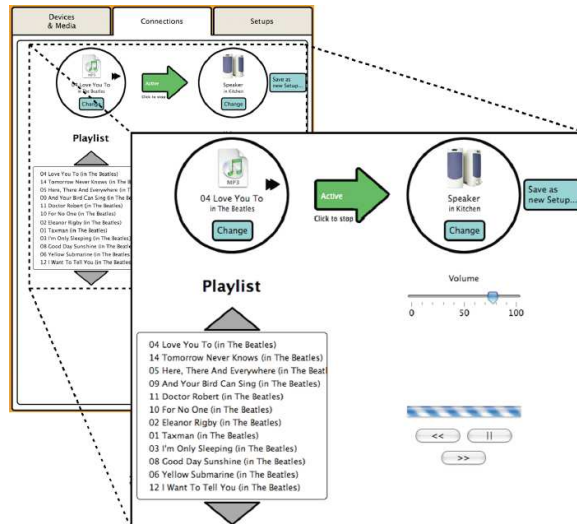
Figure 2.2: *OSCAR Configuration Screen*

composition with other components to create a whole functional software program or system. The idea of software components was first introduced by McIlroy [154] and subsequently introduced into the Unix operating system as "pipes" and "filters" which can be used to connect the input and output of software programs.

Components themselves are entirely written in software but can be very tightly coupled to hardware components - such as hardware based user interfaces. This blurs the edges between where a software component ends and where an associated hardware device begins. From the viewpoint of a system that uses components this boundary is largely immaterial as many software components cannot be decoupled from their hardware devices - an example would be device drivers; the hardware is useless without the drivers and the drivers are useless without the hardware. As these can be so tightly coupled it is legitimate to consider the functional coverage of a software component to include that of its associated hardware devices.

Components interact with each other by establishing bindings; bindings can be either *first-party* or *third-party*. In a first-party binding a client component locates the component it needs and establishes the binding itself. Dynamic resolution of configuration of first-party binding, such as dependencies, has been extensively researched [48, 107, 120, 167] and are not specifically addressed here. More interesting from the viewpoint of the work presented in this report is the concept of third-party bindings where the decisions of which components to be bound is decided by, and established by, a third-party component [82]. Third-party binding allows components to be bound together by a configuring component -

a pattern that will resurface later in this work.

2.3   Describing Configuration

The previous two sections discussed the types of configuration and a selection of types of interactive system that can be configured. This section will continue this strand by discussing the ways in which configurations can be described by a user. This section makes the distinction between process and product; while there will inevitably be some discussion on the product generated by a configuration technique (a configuration described in some suitable format) this section will concentrate primarily on the interaction techniques available in the process of manipulation of the configuration and their mode of use.

2.3.1   Configuration Files

Configuration files are the historical method of specifying configuration settings within a computing environment and date back, at least, as far as Unix configuration and *dot files* [216] and MS-DOS *config.sys* and *autoexec.bat* [155].

Configuration files can be regarded as a simple human readable database of key, value pairs that may have additional structure imposed by delimiting the file into sections or hierarchies. Several de facto standards have emerged including INI files [37], the Windows Registry [84] and a general trend towards XML [156].

Since configuration files were designed to be human readable the simplest technique of editing a configuration was to open the file in a text editor and edit the values manually. This would often be performed with the appropriate documentation to hand - in some cases entire books were devoted to the intricacies of a software application's configuration files [63].

GUI applications were later fitted with integrated interfaces to the configuration files to assist a novice user in making configuration changes as demonstrated by Microsoft Office in Figure 2.3.

Due to the proliferation of configuration files some solutions have been presented to offer techniques to manage sets of configuration files [184] as well as providing integrated user interfaces for manipulating a central set of configuration files which all applications can use such as the GConf [92] editor shown in Figure 2.4.

GConf is a centrally managed configuration file store designed so that single settings can be
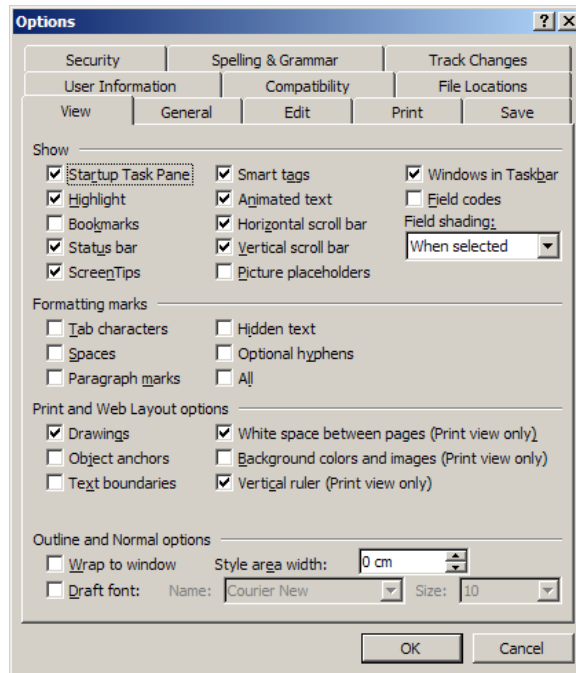
Figure 2.3: *A configuration menu from Microsoft Office 2003*
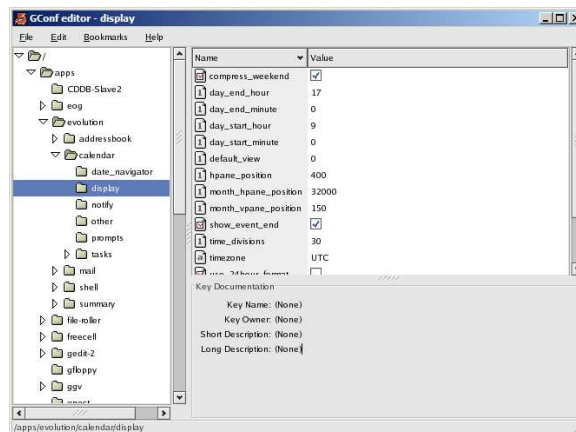
Figure 2.4: *GConf editor*

used by multiple applications - allowing a user to change settings in a single location and have it apply across every application that uses GConf.

Configuration file based interfaces are manually driven by the user who must recognise that a change is to be made and seek out an opportunity to make the change - either by finding the appropriate option in the configuration files or menu themselves or by consulting the documentation to locate where the choice of option is. Configuration files are well suited

to performing customisations but the use of simple key, value pairs means that they are unsuitable for describing more complicated relations which are necessary for describing interactions or bindings between components effectively.

Configuration file interfaces vary in ease of use and technical skill required by large amounts (compare Microsoft Office and GConf interfaces) and for textual based configuration files may require significant investment of time and effort to decipher the correct syntax and available options to make changes to the configuration.

2.3.2   Architecture Description Languages

As configuration files were unsuitable for describing software architectures a number of Architecture Description Languages (ADL) emerged for the purpose of supporting system composition and evolution [179].

A common feature within ADL's is the recognition of component based models where services or interaction techniques are bundled as modules within the system with a language used to connect the modules together. A simple filter pipeline is reproduced from the Darwin ADL [134] in Figure 2.5.
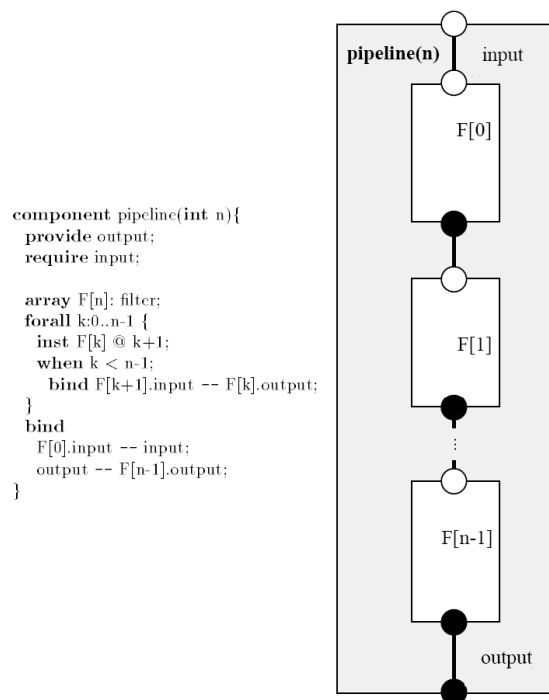
Figure 2.5: *A pipeline expressed in Darwin notation*

The main abstractions used in Darwin are components and connectors. Components are modules which abstract away the algorithmic complexity of that components functionality while connectors are used to build a logical structure out of the components. Components can be composited together with connectors to create larger logical components. Darwin supports dynamic instantiation of components which allows for new components to be added at runtime however access to the services these dynamically instantiated components provide is restricted since each instance cannot be modelled within the description language.

Many such ADL's have been developed [12, 182, 204] to meet academic and commercial requirements. The syntax for ADL's tends to be complex but may be supported by GUI based tools to facilitate their use. Due to the static nature of a ADL based configuration they are of most use when describing static structural architectures. ADL's offer minimal support for dynamically changing configurations or for context aware selection of components.

ADL's can have different levels of ease of use and may be supplemented by a GUI to make the process of describing a desired configuration simpler and easier, but due to their static nature they are unsuitable for reasoning about changing or evolving systems.

2.3.3   Component based editors

The previous section covered architecture description languages which, although they may have GUI support, are designed primarily with the ADL syntax and structure in mind. This section will discuss component based editors which are primarily designed with the GUI in mind although they may be supported by an ADL or similar construct internally.

Component based editors are often used for Rapid Application Development (RAD) by software developers [25, 97, 218, 224] but the focus of this work is on users thus three exemplar component based editors which are designed for use by end users, rather than developers, are discussed.

Max [180] was originally developed in the 1980s as a graphical programming environment for music and multimedia. Max is a modular design and allows new first and third party components to be added to the software by users. Max is often used with the Max Signal Processing (MSP) extension package which users use with the Max graphical interface, shown in Figure 2.6, to add components and connections to manipulate digital audio signals in real time to create their own personalised synthesizers and effects processors. Add-on packages for the Max engine are available from both Max developers [181] and third-party developers [22] to allow Max to process real time video which demonstrates the flexibility

of their component oriented approach. Max has become the *lingua franca* for practitioners of computer based music performance due to its accessible and modular structure [175].
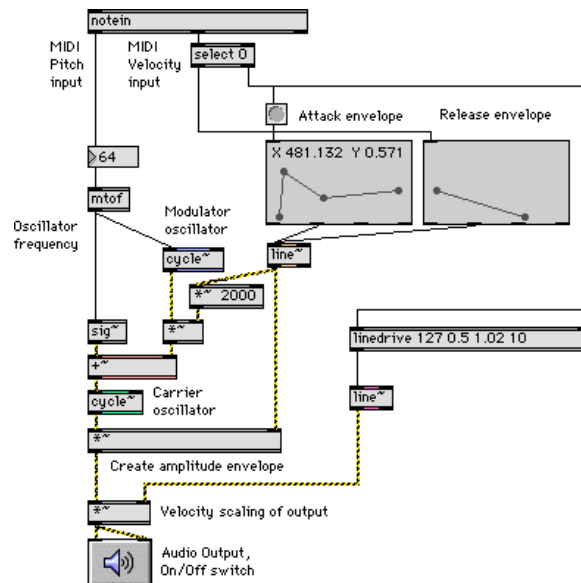
Figure 2.6: *Max/MSP*

Another component based editor is Speakeasy [162] which is a web based user interface designed to allow end users to configure and assemble devices of interest to them. This is a particularly relevant example as it is in resides within the domain of ubiquitous systems and presents a way of allowing users to configure an interaction. Speakeasy offers task oriented templates as well as connecting components together directly as shown in Figure 2.7. Task oriented templates are of the "form-filling" style and allow a user to essentially fill in the blanks in a template. The example Newman offers is the "give a presentation" task which has fields for the user to enter a file and projector to use. Speakeasy is aimed towards tasks that will have an immediate feedback in the environment.

Jigsaw [109] is a lightweight configuration service based around component based graphical editing. The Jigsaw interface, as demonstrated in Figure 2.8, is composed of a list of available components and a canvas for viewing and connecting components together. The components are represented as jigsaw pieces which offer the user insight into the function of a component (input, filter, output). When a jigsaw piece is selected then any other pieces that it cannot be connected to are faded out to allow the user greater understanding of how the components can be connected. Jigsaw draws strength from its simple interface which allows users to composite smaller devices together to model their desired behaviour.

As demonstrated by the images of component based editors designed for different uses
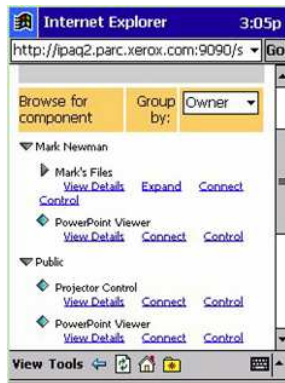
Figure 2.7: *Speakeasy*
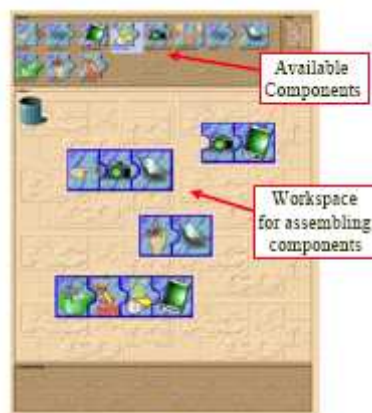
Figure 2.8: *Jigsaw*

within this section (Figures 2.6,2.7,2.8) the difficulty of using a component based editor can be radically different based on the complexity of the underlying model that needs to be exposed to the user. For example, the Max/MSP system supports a wide range of ways in which components can be connected and this complexity is inherent in the interface design, while the Jigsaw system exposes a much simpler model with a maximum of 2 connections per component resulting a comparably much simpler interface.

As with configuration files and ADL's; component based editors are driven purely by the user and used to generate static structures. If the circumstances or context of a situation change the component based editor must be used to change the configuration.

2.3.4 Automatic Configuration

Section 2.2.4 discussed component based systems with automatic first party bindings, this section will address systems which provide for fully automatic configuration using third-party binding.

The basic idea of an automatically configured system is that you can devise the best connection by using a single *utility function*, as presented by Sousa [213]. Sousa claims that such a utility function is responsible for capturing (i) the preferences of the user, (ii) the preferences of which devices or services are better at supplying the required service and (iii) the quality of service levels and preferred trade-offs.

Much of the work done for automatic configuration in ubiquitous environments is focused on architectures to mediate conflicts between the users preferences and the environmental preferences and to perform policy resolution in the event of detectable conflicts [40, 234]. These systems treat the users preferences as a well-formed set of rules which can be implemented in advance to cover every eventuality - in reality this is rarely the case.

The result of automatic configuration is that the system does not rely on the users input to configure it - but at the same time automatic configuration cannot benefit from user interaction either. Norbistrath and Mosler [164] recognise that automatic configuration will never be reached due to the human factor involved in specification of desired services however their approach is to shift much of the configuration work to the development phase of the system rather than offering full control of the configuration to the user.

2.3.5 Recommender

Some exemplars of the *recommender* style configuration tools which represent a particular middle-point between manual and automatic configuration are presented here. Recommender tools monitor the previous history of the current user, and sometimes other users for collaborative versions, to determine patterns which might indicate which configurations are more successful. If a recommender algorithm is able to identify what it believes constitutes a more successful configuration than what is currently in use then it can recommend this to the user, who can choose to accept it or remain with the status quo, or can automatically update the configuration to the recommended option, or it can be used to filter out unsuitable configurations from a list of configuration options presented to the user.

Goldberg et al. [94] introduced the first such system, Tapestry, using the term collaborative filtering. Tapestry is a mail system where users could specify filters which worked

collaboratively. Filters could reason on the results of other users actions on documents - allowing users to form queries such as only highlighting articles that another user had marked as interesting. Filters written by one user can be used, and extended, by other users.

Resnick and Varian [185] use the more general term *recommender system* instead of collaborative filtering, a style adopted here. Resnick and Varian point out that recommender systems may not explicitly collaborate with specific users - data may be aggregated, and recommendations are not restricted to filtering information they can suggest particularly interesting items as well.

Examples of recommender systems include Amazon's product recommender [126] as well as many more commercial collaborative filtering and recommendation systems [198]. The Domino [19] framework will briefly be examined here and is a recommender system that is of more interest within the ubiquitous computing domain.

Domino is a component based collaborative system where components can be incrementally added to create an evolving system. Components can be discovered through interactions with other domino systems on the same network segment and these new components can be added in runtime. Domino can exchange usage data about the components that can be exchanged to provide the user with an indication of how useful others find this component. This usage data can serve to recommend particular components, or combinations of components which the user might themselves find useful. Recommendations for a user are found by comparing the components in the users current configuration against configurations in which new components have previously been frequently used. Once a recommendation is suggested to the user it is up to them to decide if they wish to accept or reject that recommendation.

Domino has been demonstrated in the form of a game "Castles" where components are presented to the user as units and buildings. Players of the game then receive recommendations on troop and building types to use based on the success of other players. Figure 2.9 shows Castles explaining why a recommendation was made.

Recommender systems represent one particular approach to the problem of deciding a configuration. The user retains ultimate control while still benefiting from some aspect of automation to reduce the amount of work they need to do to configure the system but still require user intervention to give good results. Recommender systems could be adapted such that their recommendations are always accepted - removing the reliance on user interaction; however, recommender systems are fallible - the process of inference across usage or popularity data means they will be bound to, eventually, recommend the wrong component or configuration.

Figure 2.9: *Domino/Castles explaining details as to why a recommendation was made*

2.3.6 Programming by Example

Programming by Example (otherwise known as Programming by Demonstration) is an approach to programming where the user inputs concrete examples which are then generalised by the system automatically.

Pygmalion [210] was the first programming by demonstration system which took the form of an "electronic blackboard". Users would edit graphical snapshots of the computation and run partially specified programs. When the program reached a branch it would then ask the user what to do next. Pygmalion was a toy system intended to demonstrate the concept and so was not suitable for large problems.

The idea Pygmalion presented inspired a number of other systems in the same space for beginner [125] and non-programmers [76]. Later programming by example systems introduced inference [142] to allow the system to infer the users desired behaviour. Further implementations of the programming by example ideas were applied to more domain specific problems such as repetitive text editing [159, 235] and graphical editing [112, 122, 124].

Programming by example could be used within the domain of configuration by allowing an interface to generalise the actions of the user into a configuration. This would still be manually driven; but, like recommender algorithms, would reduce the effort required to program the configuration.

2.3.7 Overview

The previous sections discussed the current methods of configuration which were configuration files, architecture description languages, component based editors, automatic, recommender and programming by example. Each style of configuration has its own types of interaction as well as its own benefits and drawbacks in different circumstances which are summarised in Table 2.1.

Style	Can be driven	
	Manually	**Automatically**
Configuration File	✓	
ADL	✓	
Component Based Editor	✓	
Automatic		✓
Recommender	✓	✓
Programming by Example	✓	

Table 2.1: *Types of configuration technique and the mode of use*

Manually driven configuration is clearly inappropriate in many circumstances; particularly where reconfiguration would be frequent. Users cannot be expected to manually reconfigure the system every time a new visitor comes to the house or circumstances change. In the pathological, case it cannot be required to perform a manual reconfiguration to select the closest audio output device every time the user changes location in the home if the aim was for music to follow you. There is a distinct need for some automatic configuration, and reconfiguration, as the context of the interaction changes. However there is a conflict here, there are clearly some cases where manual configuration is entirely appropriate - such as a change in the users preferences.

Sousa [213] argues you can have a single utility function to arbitrate the automatic configuration; however this is a viewpoint that is disagreed with in this thesis. A single utility function can never be rich enough to capture all circumstances and different aspects of context will matter more or less in different circumstances. It is argued here that each of the three features of Sousas utility functions (preferences of the user, details of which devices or services are most suited and the quality of service levels and trade-offs) are actually separate utility functions that need to be weighed against each other and that there are many more possible utility functions which would represent context sensitive aspects that would need to be taken into account. Additionally the set of useful utility functions will be different for individual users.

The missing piece in this area is a framework which allows any and all of these techniques to be used if, or when, they are most appropriate. A potential model of this framework based around the idea of multiple utility functions in Section 5.

2.4   Supporting Change

So far this chapter has discussed types of configuration, what to configure and descriptions of configurations from the user point of view but has not yet addressed architectures and systems which support change that would be necessary for the idea of evolution of configuration to be realised.

2.4.1   Plasticity

The term "plasticity" was introduced by Thevenin and Coutaz [217] who explain that it is inspired by the property of materials expanding and contracting under continuous usage - the capability to withstand variations in its environment. The model of plasticity is that a plastic user interface should be able to be adapted. Adaptation can be performed either on the users explicit request or as an automatic process.
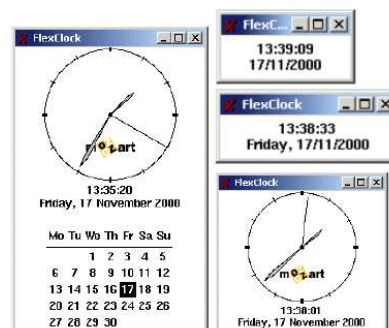
Figure 2.10: *Flexclock*

Plasticity is therefore the property of being able to make these adaptations without breaking any currently running tasks which would be necessary in a ubiquitous environment. A simple example of plasticity in action is the FlexClock [101]. FlexClock changes its output style based on the dimensions of the screen real estate it has been given - ranging from a simple HH:MM format to progressively more complicated structures; including calendars. A small sample of FlexClocks representations is shown in Figure 2.10.

Thevenin and Coutaz explain that the adaptation space includes the means allowing adaptation, the target to be adaptation to, the temporal dimensions of adaptation as well as the actor causing the adaptation as shown in Figure 2.11.
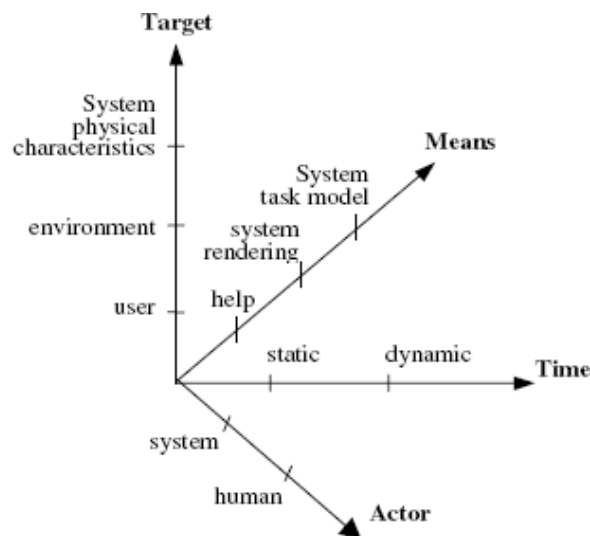
Figure 2.11: *Adaptation design space*

The means axis represents the ability of a system to support adaptation and change and the extent to which adaptation can occur. The system task model represents the systems implementation of what the user wants to do and is responsible for orchestrating interactions from beginning to end while the rendering techniques are the observable behaviour and interface of the system.

The target axis denotes the objects which the system can adapt to. The most obvious of these would be the user, or users, themselves but a non-exhaustive list would include adaptation to the physical environment, location and interface characteristics.

The temporal axis represents the time aspect of adaptation and deals with when adaptation occurs and may range from statically between sessions or dynamically at run time.

Finally, the actor axis is the initiator of the adaptation. Adaptation may be triggered automatically, due to a change in circumstances or context, or may be explicitly signalled by the user of the system.

Using these dimensions a categorisation can be made of the above axes into the "how" (means), "what" (target), "when" (temporal) and "who" (actor) of adaptation which will now be discussed in more detail.

2.4.2 Means of adaptation

This section explores the software components of a system involved in adaptation and the abilities it has to facilitate adaptation which represents the "how" of adaptation. A common model for this is to separate the task model controlling the interactions from the domain representation of the data and the rendering components used. This section will discuss a number of task models which handle this separation followed by rendering environments designed to support plasticity as well as architectures intended to support change.

2.4.2.1 Task Models

Task modelling is the act of describing aspects of an interaction with a computing system. Task modelling aims to represent features such as the logical structure of a task or meta properties about the users performance during the task in order to describe the goals and processes of the task independently of the interaction methods.

The GOMS (Goals, Operators, Methods, Selection Rules) model was introduced by Card, Moran and Newell [33] and has subsequently spawned a large number of varieties [116]. The GOMS family consists of successive decomposition of task goals into subgoals which are composed of methods and selection rules. GOMS is predictive model and can be used to assess task costs but does not explicitly deal with temporal properties of tasks.

Task tables are an approach which primarily adds feedback to task descriptions. An example of this is UAN [106] (User Action Notation) which is a behavioural representation of the task. Task table based methods are best for representing the connection between a users action and feedback generated from that action and while they may offer some support they do not specify temporal relationships between subtasks well.

Task trees make the temporal relationship of subtasks more explicit by imposing hierarchical structure. ConcurTaskTrees [169] are a typical example of a modelling technique which uses the task tree approach. Subtasks are composed together using operators which specify the subtasks temporal relationship in terms of features such as interleaving and synchronization. This allows modelling of the temporal properties of subtasks and the imposition of order in a task.

This is not an exhaustive list but merely serves to illustrate the types of task modelling that can be performed. Many different types of workflow management (UML activity diagrams [62] and Petri Nets [172] for example) can be adapted for task modelling. Combinations of techniques can be used together to model different parts of the task.

Since these techniques are, more or less, formally defined they can be used to reason about the overall task structure and properties when interacting with the user. The task models themselves are not the focus of this research; but the concept of separation of the task modelling from the rest of the system is a critical concept in some of the systems discussed in the next section.

2.4.2.2   Supporting Change

To continue the theme of how to deal with adaptation, approaches which have explicit support for change are examined. This section will cover the major frameworks for plasticity and adaptation with the ultimate aim of demonstrating the lack of generic decision support for how an adaptation can occur - something that is essential for the idea of evolution.

The first approaches considered are the *Arch* and *Slinky* models [16] which provide a functional top down view of the interaction comprised of 5 key components shown in Figure 2.12.
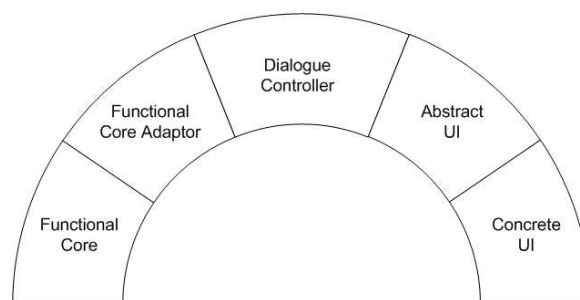
Figure 2.12: *Arch Model*

In the arch model the functional core represents the domain specific component which is responsible for controlling, manipulating and retrieving domain data and performing any other domain related functions.

The dialogue controller is responsible for task modelling within the architecture and has the ultimate responsibility of sequencing subtasks and performing data transformations between the domain formalisations and the user interface formalisations.

The abstract and concrete UI's represent the user interfaces used in interactions. The abstract UI acts as a mediation or presentation layer between the dialogue controller and the concrete user interface which is the actual interaction object used by the user. The

abstract UI may provide concrete UI independent objects for use by the dialogue controller - for example a "select from a list" object which may be implemented as either a menu or radio button within the concrete user interface.

The final component is the functional core adaptor component. This is a mediation component between dialogue controller and functional core which is responsible for triggering domain initiated tasks (via the dialogue controller) as well as providing any domain oriented tasks that the functional core does not provide (such as aggregation or ordering of data).

Slinky Metamodel
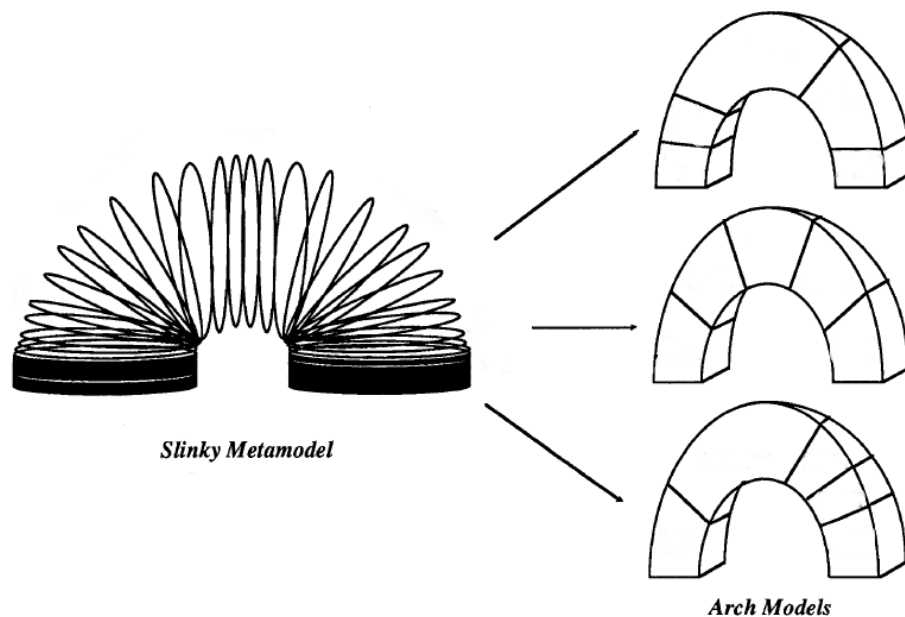
Arch Models

Figure 2.13: *Slinky Meta-Model*

The slinky metamodel as shown in Figure 2.13 is a generalised view of the arch model where emphasis can be shifted from component to component. The slink metamodel derives its name from recognition that the simple arch model is insufficient to fully capture the need for varying amounts of functionality in each component of the model. This allows emphasis to be shifted away from the functional core in architectures emphasising interaction while architectures emphasising data can instead emphasise the functional core.

Speakeasy [67] is an approach designed to allow devices and services to interact with little prior knowledge of each other. Components use small fixed domain-independent interfaces and mobile to realise this. Mobile code is the ability to transmit executable code across the network in order to extend the functionality of a device. A simple example, shown in Figure 2.14, is that a projector may have a "control panel" component which can be

instantiated on device with a GUI to control the status of the projector (on/off, active input ports). There may be multiple versions of the control panel available for different contexts such as PDA, Desktop PC with different capabilities. This approach is known as the *recombinant computing* approach.
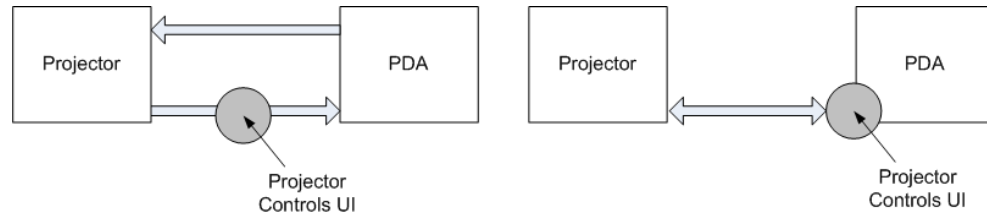
Figure 2.14: *Speakeasy example where a Projector sends a control panel to a PDA using mobile code*

SUPPLE [231] is an approach which includes the ability to adapt to device characteristics by rendering a GUI display generated at run time within screen-size constraints. SUPPLE uses a utility function which assigns "costs" to each widget representing the "ease of use" and then composes a widgets to perform the required tasks together within the given screen size constraints to create a working display with the minimum cost. SUPPLE is, however, limited in that it can only reason about the quality of a widget by comparing costs between respective widgets.

Comet [32] (COntex of use Mouldable widgETs) is an application of plasticity applied at the widget level. Comet's are introspective components which publish quality of use guarantees for a set of contexts of use. The Comet architecture supports adaptation by polymorphism (change the form of a Comet), substitution (replace a Comet), recruiting (adding new Comets) and discarding (removing Comets). These adaptations are triggered by policies; at which point the current context of use will be derived and compared against the quality of use guarantees published by available Comets and the Comets updated appropriately. The disadvantage of this approach is that each Comet must be able to identify its own quality of use statistics in each of the contexts of use it is likely to appear in which will be impractical for large numbers of contexts (or unions of contexts).

Crease et al. [46] presented a system where the configuration of output sources was controlled by a "modality mapper" service. The modality mapper was responsible for deciding which modalities to use based on the feedback type and used a weighting system to decide which modality, and ultimately which concrete output device, should be used.

The Cameleon [11] reference architecture, as shown in Figure 2.15, is split into three levels

of abstraction - the Interactive Systems Layer, the Distribution-Migration-Plasticity layer (DMP) and the platform layer.
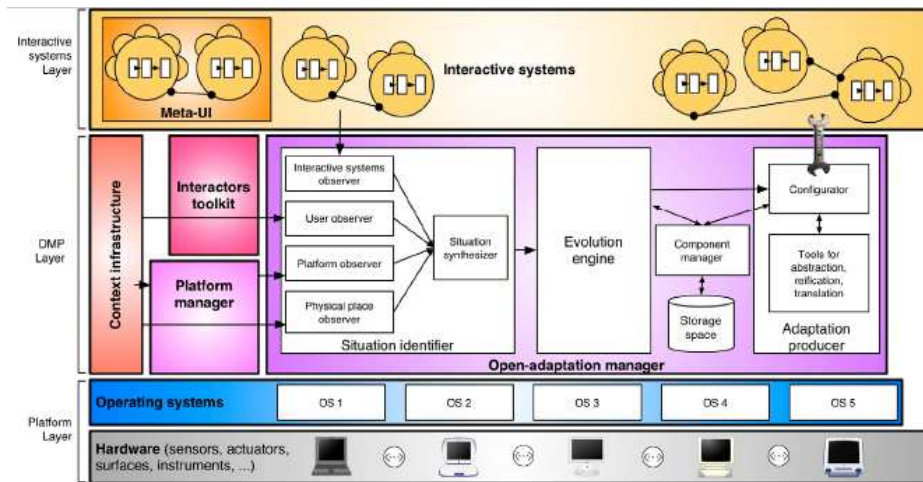
Figure 2.15: *Cameleon Reference Architecture*

In the context of this work, the most interesting aspect is the DMP layer which contains an *evolution engine component*. The evolution engine component in the Cameleon architecture is notified of a change in context by the situation identifier component and is responsible for responding to this. The evolution engine then identifies any UI components that must be replaced or added and notifies the configuring component which enacts the changes.

It is therefore the responsibility of the evolution engine to handle the configuration of the system. In the CamNote system built upon the Cameleon architecture the evolution engine is implemented as a rule engine of the form "if a new PDA arrives, move the control panel to the PDA".

USiXML [223] (USer Interface eXtensible Markup Language) is a system for designing user interfaces for specific contexts of use built upon the Cameleon reference architecture. The user interfaces generated from USiXML are then transformable using a rule set to enable changes from one context of use to another.

The Cameleon reference architecture does not specify a generic approach to implementation of the evolution engine; rather it is expected to be implemented using existing techniques such as policy engines or rulesets to choose the correct configuration to use in the current context.

The approaches described above are all similar in that they have generic structures for describing an interaction or combining widgets or components together to create an

37

interaction but do not have the same facilities for generic methods of actually choosing which of these configurations to use. Many use user specifiable or designer supplied rules and weighting schemes to make these decisions but these approaches are not generic and do not allow the full range of reasoning about change.

2.4.3   Target of adaptation

The target of adaptation covers the range of interaction devices that can be adapted to and the criteria that can apply to decide which interaction devices should be used. Some of these criteria could include; the range of available devices, contextual information about the user and their environment, social context, policies or preferences which have been set up by the user.

Thevenin and Coutaz describe the canonical targets of adaptation as (i) the system physical characteristics, (ii) the environment and (iii) the user. Each of these targets implies one or more models existing for each of the targets which can describe the current, or expected, condition of the target. This allows a system to make choices about the appropriate configuration based on knowledge of the environment within which it is placed.

There are a variety of sources from which this information could be obtained. It could be obtained directly from the user via direct interaction (GUI button with "Leaving house" pressed) or implicitly inferred via one or more sensors operating continuously (door sensor detects someone leaving). The model could contain static elements (the bathroom is located across from the bedroom) as well as dynamic elements (current temperature or light levels) which can be combined together during the reasoning process.

This axis of adaptation in particular is elaborated upon further in Section 6 within the context of the model proposed in this work and an additional axis proposed which separates the target of adaptation from the source of the information.

2.4.4   Actor of adaptation

Another axis of adaptation is that of the actor; that is who is responsible for triggering an adaptation and who is responsible for the resulting configuration. In this section the term actor refers to the entity responsible for triggering or otherwise requiring a change in the current configuration while there may be additional users who are stakeholders or targets of adaptation but who did not trigger the reconfiguration.

There are two main categories of actor who will interact with the system and trigger changes or identify new opportunities for adaptation; these are human and computational.

Users can (i) provide inputs at design time - prior to creation or use of a feature (e.g., preference files read by a function), (ii) interact with the selection process directly as part of the evaluation process, (iii) indicate a changed opinion thus triggering a re-evaluation or (iv) interact implicitly, in which some computational entity gathers usage information or indications of the user's satisfaction over time to determine how to change configuration. An example of a system which uses direct manual adaptation by a user is Jigsaw [109] where the user composes jigsaw pieces together manually to create a functional system of their own design.

This is complicated by the presence of multiple human actors who will be stakeholders in the configuration of a system. Reconfiguration could be triggered by a diverse set of human actors. Each of these human actors may have very different ideas about how the system should be ideally configured resulting in disagreement over the configuration which must be explicitly modelled to be dealt with.

McGee and Gray [151] provide examples of circumstances which result in conflicting aims; including shared interaction spaces, multiple care conditions and volatility of behaviour and beliefs. They go on to argue that to cope with the identification, negotiation and resolution of conflicts in a home care system requires a solution that may be a combination of socially or clinically negotiated aims that may be implemented at a system level. Such a framework of conflict negotiation is discussed in Section 4.

In addition to human actors there will be a number of computational entities which are responsible for triggering configurations and may be responsible for a configuration result. These computational agents assist in configuration by allowing rapid or automatic reasoning about available configurations which may be difficult, impractical or impossible for a human agent to perform. This may be used to prevent configurations which are recognisably unsafe or invalid as well as automatic selection of configurations which have been previously marked by users as being preferred. Examples of systems which involve a computational agent in the interaction decision making process are those which use a utility function such as Sousa [213] or those which involve an automatic contextual decision making element such as FlexClock [101].

Like human actors there may have multiple computational actors in the system at the same time and like human actors these may conflict with each other. For example there may be a computational actor which attempts to configure the system based on the users location and another which prevents particular interaction devices from being used at particular times of

the day. Further extending this there may be conflicts between computational and human actors. An example of this may be when a user attempts to perform some action which is prohibited by a system policy. The "correct" resolution of this may differ depending on the circumstances; in some cases (such as low priority alerts) the policy may override the users intention in order to preserve some greater goal (such as no noise after a certain time) but in other cases (such as a high priority alert) the human intention may override the system goals (such as in emergency conditions).

Some actors may trigger configurations based on interaction with other actors or on behalf of other actors. An example of this would be a computational actor which interacts with a user or multiple users (themselves actors influencing the configuration) before it itself triggers configurations.

In general a configuration may be a joint responsibility between both human and computational entities where the triggering or resolving of a configuration is the result of interaction between multiple actors.

2.4.5   Temporal adaptation

This axis investigates the temporal aspects of adaptation; i.e. when to adapt a configuration. Here the discussion briefly explores the different points of time that configurations can be made and the different temporal techniques that can be available. This section generally refers to the actor responsible for the configuration as human but the following discussion applies equally to computational agents.

Two broad temporal approaches to interaction are identified. The first one is one-off or sporadic interaction where the user specifies their needs and wants at the time of configuration with no plans to change it. A good example of this is the definition of medical protocols, such as Wisecare [98] protocols. The Wisecare protocols define a set of policies which are created a priori of system implementation and deployment, These policies are fixed by current research and are essentially unchanged throughout the life-cycle of the system. Although the protocol may be improved through later revisions the intention of the policy is that it is not user modifiable and is fixed once created.

The second type of interaction would be continuous or regular interaction where the user frequently interacts with the system, or plans to interact with the system, to assist in the evolution of choice of suitable interaction techniques. A particularly relevant example is provided in the work of Fickas [74] where the user is introduced to a system in a series of stages or goal attainment phases designed to assist with cognitive rehabilitation. In Fickas'

work the interaction with the user is expected to be revisited multiple times throughout the life-cycle of the application where it can be adjusted to be most suitable for the users current state of development in using the tools.

In addition to these two temporal frequencies there are two modes of temporal interaction identified. In the first case a configuration is started and completed within a single, potentially atomic, action. This is analogous of most automatic approaches to configuration which execute to completion within a single unit of time. The alternative approach is to allow configuration to be interrupted or deferred during the configuration. This allows for situations where a configuration is requested of a user or a computation entity but which the actor is not capable of accommodating the configuration request at that point in time. The actor may wish to delay their decisions until a later point in time (either because of lack of data or simply because they may be unavailable).

2.5   Overview

This chapter has presented different types of configuration and their relationships with each other and introduced the concept of evolution which underpins the ideas presented in the rest of this work. Systems that the concept of evolution can be applied to are identified, followed by techniques used to describe configurations in the existing literature. Finally, existing approaches to supporting configuration and supporting change over time are explored and the different aspects that such systems can adapt to are examined.

Throughout this chapter a number of systems which can do one specific method well have been presented but there is no generic approach which would allow combinations of these techniques to be used at the users discretion. The next chapter presents a study conducted to determine the necessity of change within the context of audio reminders in the home.

3

Configuration Evolution in Multimodal Interaction - A Case Study

> **Published Work:**
>
> This section incorporates material that has previously been published as *Audio Reminders in the Home Environment* [152].
>
> My contribution in terms of this work was to jointly devise the experimental protocol, aims and objectives of the trial as well as implementing the software that would be required to undertake the trial.

As a part of the MATCH project and as a pilot for some of the ideas being generated for this thesis - a study was carried out to explore performance of, and preference for, different types of auditory reminder the home context. This study compared three different types of audio reminder in a home setting. These were earcons, speech and a pager sound. The purpose of this experiment was to show that there is a genuine need for personalisation and reconfiguration of home care interactive systems within the home through a motivating example. This experiment measures user preferences between each of the three different audio reminder types as well as measuring their performance as reminder notifications This work was carried out by myself, Marilyn Rose McGee-Lennon (University of Glasgow) and Maria Wolters (University of Edinburgh).

Since there have been very few studies comparing speech and non-speech audio presentation, it is very difficult to posit general design guidelines or to determine which approach should be used in different circumstances. This chapter presents a pilot study designed to address this gap in the literature.

3.1 Audio reminders

This chapter discusses several ways of delivering audio reminders such as pager-style sound alerts [111, 236], earcons [51, 52] which are sequencing of sounds with meanings, and spoken dialogue systems [176, 177]. Simple sounds alerts are often used in pager or notification systems and can be used to notify people with minor cognitive impairments to scheduled or unscheduled tasks or events and have been shown to be effective in trials [236]. In the comparison, the effectiveness of the three audio options is compared and their acceptability and the degree to which they interfere with the user's current activities is investigated.

Earcons consist of brief, structured sounds which map onto specific meanings [23, 26]. Sainz de Salces, England, and Vickers designed and tested a range of earcons for alerting older people to events in the home [52]. Their earcons contained two structured pieces of information; one denoting an appliance and another denoting that appliances status. Older users in this study found earcons difficult to remember and suggested changing them to auditory icons [88] which are based on sounds from the everyday environment (for example bird calls) rather than abstract sounds as in earcons; however users typically report auditory icons to be annoying after prolonged use [189,207]. Another study [189] found that earcons are easier to map to application tasks than auditory icons.

Lines and Hone [127, 128], investigated the use of speech audio reminders within home care systems which was implemented in the Millennium Home care system [171]. They used speech over loudspeakers to deliver critical alerts to ensure quick and reliable alerts regardless of the users position. Spoken reminders have been used to deliver notifications via a nursing robot [177] as well as part of more sophisticated planning system [173] which takes the users daily routine into account although can pose privacy problems when people other than the intended recipient are present.

Exploration of the existing literature found very little research intended to compare speech and non-speech audio presentation are suitable and appropriate in different contextual circumstances to deliver the same content.

Bronstad, Lewis, and Slatin [28] compared two ways of indicating the presence of a hyperlink in a screen reader, a tone and the spoken word "link" and found that participants made fewer errors when the link was indicated by a simple tone.

Fröhlich [83] compared different audio cues used to indicate waiting time in a dialogue system, including speech, natural sounds, and musical pieces. Speech was rated the most pleasant and appropriate option, closely followed by musical indicators.

These two results indicate that the choice of delivery mechanism is dictated by the task and the audio options available. It is important to investigate the intrusiveness of different audio cues which is a key problem in sonification [197]. There is a significant amount of work in determining the effective presentation of audio alerts in demanding workspaces such as airplane cockpits [13] and disruptiveness of different types of audio stimuli [14]. The cause of disruptiveness here is the Irrelevant Sound Effect - attending to spoken utterances makes it harder to remember other items [39]. Salamé and Baddeley [192] argue that the effect arises because all speech, relevant to the current task or not, is processed in a phonological store of limited capacity. Jones, Madden and Miles [117] proposed that the interference occurs because of parallel processes of seriation, one process which maintains the order of the material to be recalled, and another that parses incoming auditory percepts for serial order. The first hypothesis implies that spoken reminders will necessarily be more disruptive than earcons, while according to the changing state hypothesis, auditory sequences with a strong serial order will be just as disruptive as speech of equal length.

Another important strand of research concerns general memory capacity. As already seen, memory may be affected in users of home care systems. Although memory tends to decline with age [10, 194], there is great variability [183]. Hence, for users with severe memory problems, explicit spoken messages may be better than earcons, whose meaning needs to be remembered. This concern is expressed by Sainz de Salces, England, and Vickers, who report that their older participants found earcons difficult to remember [52]. Vilimek and Hempel [226] found that reaction times were longer for earcons, where the mapping from audio to meaning may not be intuitive, than for auditory icons and keywords. This result indicates that good design of non-speech auditory cues is crucial.

This all points towards the need for a solution that offers multiple alternatives and a system that allows configuration and evolution based on users needs and preferences within a changing environment.

3.2   Design and Hypotheses

In this study three different types of audio reminder were compared by simulating a real life home situation where the user is prompted to adjust the setting of common household appliances. Since users would typically be engaged in other tasks when they receive these types of alert this was simulated by employing a background task that would be running concurrently with the presentation of the audio reminders. The task chosen for this was the digit span memory test [41] which would be highly sensitive to Irrelevant Speech effects because it is a serial recall task. The three reminder types were compared along both the dimensions of user performance as well as user preference.

User Performance was defined as "The best type of reminder is one that distracts the user the least while still enabling him/her to successfully perform the required action" and was measured as a combination of errors made in digit span recall and by number of correct responses to the reminder task. During the experiment, performance was measured for both the background task (digit span) as well as the primary task (adjusting appliances). The user performance dependant variables were defined as:

Digit Span Correct: This was scored 1 if a subject successfully repeated a digit span with all numbers in the correct order at the correct position, 0 otherwise.

Reminder Correct: This was scored 1 if a subject selected the correct appliance (TV, heating, fan) and the correct action (up vs. down), 0 otherwise.

User Preference was defined as "Regardless of performance, users will show clear individual preferences for reminder types. These preferences may depend on contexts of use". Preference was measured subjectively via interview and questionnaire.

The main independent variable was **reminder type** (Speech, Earcon, Pager). The following predictions were made:

H1: (Performance) Participants will make more errors (wrong appliance selection) attending to reminders when presented with earcons, because speech gives explicit instructions and earcons do not and the pager sound will force participants to check the full reminder instructions textually.

H2: (Performance) Speech will result in more errors in participants' performance in the digit span task than earcons or a simple pager sound due to irrelevant speech effects.

H3: (Preference) Participant will report a preference for shorter reminders (earcons, pager sound) rather than longer ones (speech).

H4: (Preference) Participants will report a preference for the reminder type that interferes least with their performance in the digit span task.

3.3   Participants and Procedure

This section explains the experimental task; to attend to an auditory reminder which was played via hand-held computer (Dell Axim). The experiment was a within subjects (N=11) repeated measures design. The three audio reminders previously discussed (pager, earcon and speech) were used to alert the users to an operation (turn up + or turn down -) that needed to be applied to a particular household appliance (heating, TV, fan). This resulted in 18 ($3 \times 2 \times 3$) different reminders. An equal number (18) of blank (no reminder) trials were included in order that a reminder was not presented on every trial. The reminders were played randomly throughout the digit span trial in order to reduce expectation that reminders would play at regular intervals - this was accomplished by "shuffling" the set of reminders and non reminders at the beginning of the trial to create a randomly ordered set which still retained the same number (36) and relative composition of trials for each participant. On a trial with a reminder present the reminders was played after the digit sequence had concluded and they could operate the control interface shown in Figure 3.1. The experiment was concluded with a questionnaire on perceived performance and preference for the different reminders. The complete experiment for each participant, including questionnaires, lasted an hour.

Eleven (11) native English speakers were recruited for this study. Three participants were aged 62 ± 2 years and eight participants were aged 27 ± 5 years. 6 participants were male, 5 were female. Both older and younger participants were deliberately included as both groups are potential users of reminder systems in the home. This sample size is sufficient for detecting large effects in user preferences and user performance with power > 0.8.

Participants were screened for hearing problems using a questionnaire where only two of the older participants reported slight problems. Memory problems were tested using the Prospective and Retrospective Memory Questionnaire (PRMQ) [45]. PRMQ scores were converted into normalised T-scores, most of which were found to be well within one standard deviation of the mean with some outliers within 1.3 standard deviations. This indicates that no participants were aware of particular problems with their memory.

The experiment was implemented on a hand-held PDA (Dell Axim X51) as it was believed that home care systems might possibly be controlled via mobile phone or hand-held computers (among other approaches).

For the pager condition, a short and simple chime was used to indicate a reminder had occurred. Users could "check" what the reminder was for using the "Hint" button, which would display the current reminder visually in text format. This button was always available to allow users to check what the current reminder task was. The earcons were designed as simple increasing or decreasing sequences of MIDI notes. The appliance to be controlled was indicated by the MIDI instrument used (marimba, clarinet, harpischord) and the action to be performed was indicated by the direction of the notes (increasing = up, decreasing = down). The TV was associated with the marimba, the heating with the clarinet, and the fan with the harpsichord. The speech reminders were produced using the high quality speech synthesis package Cerevoice [9]. Speech messages were designed to be brief but polite. The voice used was the Cerevoice Scottish female voice "heather".

Prior to the experiment the user was shown the control interface and played each of the possible reminders they would hear during the experiment. For the earcons, each user was asked to guess the message until they got each type of reminder correct at least once. Participants were given the digit span test in the same format as they would receive it during the experiment. The digit span was increased by one each time they got the sequence correct and was repeated until they got a sequence length incorrect twice. The maximum successful digit span was recorded for each participant (N) and used in the main experiment - ensuring that the digit span used in the test was difficult enough to be cognitively demanding and not so difficult that the user could not perform the task.

During the experiment, the digit spans were visually presented to the participant with a maximum length of N and a minimum length of N-3. This reduced expectation for a fixed length of sequence so that the participants would have to concentrate on both the digit sequence as well as the reminder instructions if they received one.

An enumeration of this sequence is as follows: After seeing the sequence of digits, to attend to a reminder users had to:

1. select the button "switch to device control" from the interface screen which displays the screen in Figure 3.1

2. (for pager trials or trials where the user was unsure how to proceed) select the "Hint" button, read the help message, and click on the help message to return to the device control screen

3. select the household appliance to control (heating, TV, fan)

4. select the operation to perform (turn up +, turn down -)

5. select the button "Return" to complete the action

After reminders were attended to the participant was asked to verbally recount the digit span that had been presented to them.
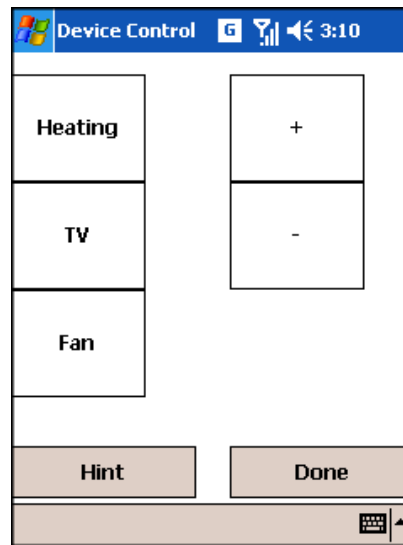
Figure 3.1: *Device Control Interface*

A post experiment questionnaire was administered to determine user preferences and rate the perceived (a) helpfulness, (b) annoyance, and (c) pleasantness of each of the reminder types. Users were asked which type of reminder they would prefer in two different contexts; alone versus with others present. This question was included because previous interviews conducted by Marilyn McGee-Lennon as part of the MATCH project in the lead up to this work had revealed that preferred modality of reminders might depend both on the content of the message as well as the context in which it is received.

This qualitative analysis is an important addition in auditory interface research as many results include error rates and reaction times without consideration for the true usability of the reminders in practice. In the home care setting in particular, it is essential that multi-modal and auditory interfaces are designed not solely for accurate and speedy responses but also for interfaces that might be usable and acceptable to users over a prolonged period within their own homes.

3.4   Results

The effect of reminder types on performance both on the primary task (attending to reminders) and the background task (verbal serial recall of visually presented digit span)

was examined. The maximum score attainable in the trials for digit spans was 36 where all digit spans (18 without reminders plus 6 speech reminders, 6 earcon reminders and 6 pager reminders) were repeated correctly. The results tables present median scores averaged across all speakers.

	No reminder	Pager	Earcon	Speech
Digit Span Correct	16	3	3	5
Reminder Correct	NA	6	5	6

Table 3.1: *Median scores for digit span and reminder task - the no reminder condition is scored out of 18 while the remaining conditions are scored out of 6*

In *Hypothesis H1*, it was assumed that speech would disrupt performance on the digit span performance to a larger extent than earcons or simple pager alerts but this is not indicated by the results: There is no significant difference in digit span scores between the three reminder conditions (Kruskal-Wallis test, df=2, χ^2=0.9636, p<0.9). Although raw scores suggest that participants tend to perform worst when presented with pager-style reminders and earcons (cf. Table 3.1), there is considerable variance in the data, as the boxplot of results in Figure 3.2 shows. The distribution of digit span scores appears to be bimodal for speech and skewed towards lower scores for the pager and earcon conditions (cf. Table 3.2).
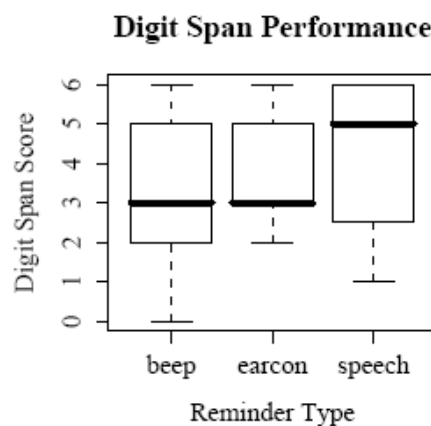
Figure 3.2: *Performance on the Digit Span Background task averaged across participants*

Low scores on the pager condition may be due to the need to use the HINT facility when presented with the pager beep. This not only involves another step in operating the tool, it also requires users to read and process a visually presented verbal message. In contrast, speech only requires auditory processing facilities, while earcons place a load on auditory

49

processing and memory. This additional load on memory may be the reason earcons did not outperform speech.

Digit Span Score	Pager	Earcon	Speech
0-3	6	6	4
4-5	2	4	3
6	3	1	4

Table 3.2: *Distribution of digit-span scores*

The table below provides data on users' reactions to reminders. As predicted in *Hypothesis H2*, reminder responses to the speech prompts are always correct. This result is statistically significant ($p<0.05$, χ^2=6.6112, df=2). Although the medians suggest perfect performance in the pager condition, the detailed results indicate otherwise. Only 6 out of 11 participants attain the maximum score in the pager condition, as opposed to 11 of 11 in the speech condition (Table 3.4). This result is partly due to participants failing to use the HINT facility in the pager condition. The two participants who performed worst under the beep condition almost never checked the action to be taken. Even though it is tempting to dismiss this reluctance to check as a fluke, we suggest that it might be even more pervasive in a field context, where the experimenter is not present, and the temptation to just guess is even stronger, because users think they know what to do next.

	Pager	Earcon	Speech
Reminder Score	6	5	6
HINT Used	6	0	0

Table 3.3: *Reactions to reminder*

Furthermore, even the two of the eight participants who always checked the box made one mistake each. This suggests that the additional step of having to look up the action is sufficient to introduce potential errors. This finding clearly needs to be investigated further, because in the home care domain it is often critical that users attend to reminders quickly and process them correctly. The results certainly suggest that the simple pager alerts [111, 236] need to be rethought.

Since the participants were not selected according to memory capacity, these analyses can only offer some post hoc insights into the role memory may play in intra-individual differences. The digit span task as it was used in this experiment mainly measures retention of information in short-term memory. Participants' performance on the pure digit span

Reminder Score	Pager	Earcon	Speech
0-3	2	3	0
4-5	3	4	0
6	6	4	11

Table 3.4: *Distribution of reminder score*

task only correlates with their performance for the pager-style reminder ($\rho=0.86$,$p<0.001$), somewhat less with performance for speech reminders, ($\rho=0.685$,$p<0.02$), and not at all for earcons ($\rho=0.483$).

The administered questionnaire first assessed how comfortable participants were with operating the reminder interface. While most people (N=9) responded that they were confident (N=7) or very confident (N=2), two participants responded that they were not confident. This indicates that in future experiments, participants need to be given more time to familiarise themselves both with the novel interface and with the different audio stimuli and their meanings.

This is further borne out by participants' assessments of the usability of the experimental interface. The most difficult aspect seems to have been selecting the device. Only 7 participants found this to be easy or very easy, whereas all participants found the digit span task and reading the screen easy or very easy. 9 out of 11 participants found the reminder facility easy or very easy to use. This may explain why some participants did not check reminders as often as they needed to. Despite this, all but one person rated the HINT facility as either helpful (N=1) or very helpful (N=10).

Any audio reminder system can only be deployed successfully if the audio reminders are acceptable to the user. Table 3.6 shows considerable variation in user preferences. The *hypothesis (H3)* that shorter reminders would be preferred was not borne out: Roughly half of the participants liked earcons, whereas the other half preferred speech.

The questionnaire yielded rich data on the reasons for participants' preferences: many people commented on the fact that speech was the easiest to get the information from but the caused the most interference with the number task. This is not reflected in the results as summarised in Table 3.1.

Users were asked about their preferences in two contexts, once when alone, and once with others present. Remarkably, the responses were similar for both contexts, except for two participants. One preferred earcons when alone, and speech when with others, the other preferred speech when alone, and earcons when with others. One user liked non-speech

audio stimuli in general (pager or earcon), while another liked meaningful auditory stimuli (earcons or speech). Since two participants preferred more than one reminder type, the totals in Table 3.6 add up to 13.

When asked for reasons for their preferences, the same feature of a reminder would be seen as both positive and negative. For example, some users felt that the explicitness of speech was an advantage when others were present, because the reminders would not need explaining to guests. Others, however, considered this explicitness inappropriate for some types of alerts, such as medication reminders. These alerts were judged as too private and should be delivered using earcons.

	Pager	Earcon	Speech
Helpfulness	3	3	5
Annoyance	1	2	2
Pleasantness	3	4	4

Table 3.5: *Median scores for each reminder type*

In the questionnaire, opinions on three aspects of acceptability were sought: whether the reminders were helpful, whether they were annoying, and whether they were pleasant. All three reminder types were rated on a five-point Likert scale. Table 3.5 presents the median scores for each property. The only significant difference was in ratings of helpfulness: Speech was clearly perceived to be the most helpful (Kruskal-Wallis χ^2=9.3553, df=2, p<0.001). The differences in annoyance (χ^2=0.693, df=2, p>0.7) and pleasantness (χ^2=1.3621, df=2, p>0.5) were not significant. Again, inter-subject variation was considerable. As Figure 3.3 shows, speech was unanimously perceived as very helpful, whereas opinions about the usefulness of earcons and pager reminders varied enormously. While the variation for speech and earcon judgements is similar, pager ratings range over the whole spectrum (cf Figure 3.4). Reasons for ratings reflected personal experience. For example, one user disliked the pager alert because it sounded like the Windows chime.

User preferences cannot be reliably predicted from a subject's digit span score: of the five participants who prefer speech or speech and earcons, only three had their highest score in the speech condition. For the five participants who preferred earcons, only two had the highest digit span score in the earcon condition. The discrepancy is largest for participants who preferred pager-style chimes: only one of three performed best with the chime.

The correlation between performance on the reminder task itself and preference ratings is stronger: all of the five participants who preferred speech performed best in the speech
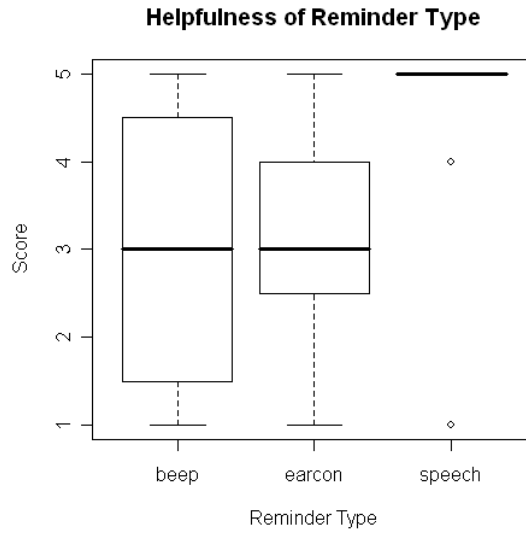
Figure 3.3: *Helpfulness ratings for each reminder type averaged across participants*
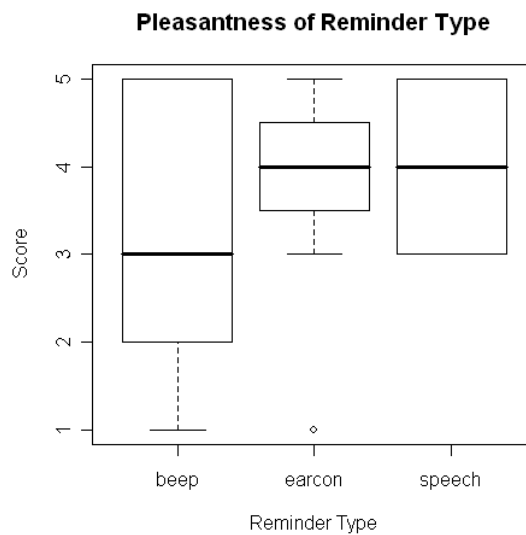
Figure 3.4: *Pleasantness ratings for each reminder type averaged across participants*

condition, whereas for earcons and pager-style reminders, there was no link between preference and performance.

These findings suggests that hypothesis H4 may have to be rejected or at least further qualified, since performance on the primary task appeared to have a greater influence on acceptability than performance on the background task. The data is too sparse to statistically

test correlations—this aspect of the experiment would benefit from testing a larger number of users. Despite this, the findings confirm that user performance needs to be carefully balanced against acceptability. If users really prefer meaningful non-speech audio, then the system designer has to create a set of sounds that are both easy to interpret and not too disruptive.

Preferred when	Pager	Earcon	Speech
… alone	3	5	5
… with people	3	5	5

Table 3.6: *Preferences for reminder types - multiple choices possible*

3.5   Overview

It is crucial that auditory reminder research continues to compare the different auditory modalities (pager, earcon, speech) rather than attempt to prove one modality as being *best* in all cases. The optimal choice of auditory reminder will depend on the task or operation to be attended to, the urgency of the reminder, the importance of correctly attending to the reminder, the degree of disruption caused by the reminder, the context in which the reminder is being received, previous exposure [8], and the users' perceptual and cognitive abilities. Some of these can be measured quantitatively (such as perceptual abilities), others need to be assessed qualitatively (such as previous exposure).

In some applications, such as home care reminder systems, the variation in all of these factors will be such that the user(s) will need to be able to personalize the system to their needs and modify these settings as their requirements change over time and space.

This research indicates that it is very important that users are capable of configuring the system as preferences are not necessarily always the same as the users best performing modality choice. Therefore, it is important to allow the choice of modalities within a home such that a user is capable of choosing whether to satisfy their preferences or to aim for the highest performing system. Likewise, it has been shown that not all users agree on which modalities are preferred as well as showing how the preferences change based on context, accordingly it is important that it is possible to configure such a reminder system such that its behaviour can be dependant on the context within the home (in this case who is present etc.). Additionally, perceived performance is not always the same as actual performance - which implies that additional factors need to be taken into account within configuration

above and beyond the users preferences.

This chapter has demonstrated the need for personalised configuration and the need for this configuration to be able to change over time and subject to the requirements of the environment. In the next chapter, the process of evolutionary configuration is discussed which allows users to adapt a system or application over time to meet their needs; while taking into account the contrasting preferences of different users and the different techniques which could be used to achieve a configuration goal.

4

The Process of Interaction Evolution

This chapter briefly discusses a number of sources of change that can affect adaptive systems before going on to discuss the process of evolution for adaptive systems from a user's point of view.

4.1   Sources of Change

Before the process of evolution for adaptive systems is introduced, it is necessary to first touch upon some of the sources of change that can affect the configuration process in order to explain the process within a suitable context. This section should be viewed as a short summary; a characterisation of the configuration space is discussed in much greater detail

in Chapter 6.

4.1.1 Stakeholders

Adaptive and configurable systems can involve multiple users and/or multiple stakeholders. Within a home environment there are likely to be partners living in the same space, friends and family living elsewhere who are involved in care or interested in its status, visiting medical personnel such as community nurses and remotely located medical staff, such as a consultant in a clinic that the patient visits [153]. Within an office or working environment these may still include friends and family that wish to stay in touch during the course of the day, co-workers, management, customers and clients as well as representatives from other companies or government agencies who may visit during the course of the day.

These people are referred to as stakeholders [203] if they have a direct or indirect interest in how the system works, how the system is used, or the data it generates or provides. Many stakeholders may need or want to come in to contact with the data or devices of a homecare system themselves directly either in the clients home or remotely. In this case, these stakeholders have to be considered potential end users of the homecare system. Stakeholders would include external agencies responsible for designing, installing, maintaining and prescribing the available equipment and/or changes in legislation or policy on how the devices or services can be prescribed and used.

It is likely that with multiple occupants and end users, and multiple stakeholders that peoples needs, perspectives and accountabilities [85, 129] will differ and in addition might change over time as the condition of the person and the possible behaviours of the systems change. A system's configuration may be acceptable for some but not for others. For example, the user may wish to have messages and alerts presented by speech, but this might be annoying, disruptive or confusing to a visitor if delivered via loud speakers while they are present. Similarly, information provided on a television might be disruptive of TV use by others in the household or it might allow private and potentially embarrassing health information to be read by others.

This can result in complex, dynamic and potentially conflicting needs and requirements and therefore methods are needed for identifying, negotiating, and resolving these changing requirements and interaction needs as the stakeholders interact with and use adaptive systems [149].

4.1.2 Available devices and service

Adaptive systems should be capable of providing implicit, multi-modal, and non-standard means of interacting to facilitate a more natural user experience. This is likely to include the use of speech and non-speech audio [152], graphical output delivered via mobile devices or digital television, gesture input and tactile output. Allowing users the choice of various modalities for different interaction tasks in different contexts is important [152]. Knowing which combination of these to use at any one time for any one purpose is not straightforward and is subject to change later on as experiences with a modality change how it is used.

New devices and services may become available purely as the person's context or location changes. Presenting information to the television for example makes more sense in the living room or a shared space than in the bathroom or an empty room and presenting information to a loudspeaker makes more sense if there is a person who prefers speech output present and there is no other audio output to that device at that time. As new devices and services become available, the system must be able to accommodate these new approaches and users offered ways to interact with these devices and/or services.

4.1.3 Changing needs and conditions

The needs and conditions, within the environment in which an adaptive system is deployed, will change over time - this is especially true of some subsections of ubicomp such as homecare systems. Homecare systems are particularly interesting as they pose some particular challenges in respect to changing needs over time.

It is common in an ageing population that the people being cared for will have a cluster of medical conditions to manage [166] - such as diabetes, strokes, asthma, epilepsy or orthopaedic conditions - some of which might interact with each other. This means that a homecare system must be capable of dealing with decisions on which rules to follow if health indicators from different conditions or symptoms are conflicting with each other. There is of course the added problem that conditions are not only multiple within one person but can be spread between the persons living within the home. Users of homecare technologies can be of any age and ability but a large number of users are either elderly, or have physical, sensory or cognitive impairments, or some combination of these factors. This results in a user group that should be offered appropriate choices of both traditional and novel methods of interacting with the technology and the information to increase accessibility for those with impairments. Offering choices of modalities and interaction is desirable and yet not necessarily straightforward to achieve. It is necessary therefore,

that homecare systems should be able to support preferences and capabilities that vary both between users and as care needs change.

4.2   Interaction Evolution

As part of the Section 2.1.4, the concept of evolution was introduced and this will be developed here. Central to this theme of evolution is the idea that design and development occur concurrently; a notion that has been explored by Fischer et al. [77]. In this work Fischer presented a set of complementary systems for designing a kitchen and discovered that the process of designing the kitchen would happen concurrently with the process of specifying the kitchen; users would frequently revisit and improve or retune their designs as they developed them. It is this style of activity that the process of evolution presented in this section is designed to support.

Given the multiple aspects of change presented in Section 4.1, adaptive systems should be able to adapt to dynamically changing requirements of: the client themselves, other relevant stakeholders and the situation of use. Allowing different users the choice of interaction methods for different tasks in different contexts is important to ensure both usability and acceptability.

Previous work has focused on dealing with short-term changes within a home environment such as context aware systems [20, 199] that react to situational changes. There is a gap in the literature of methods for supporting longer term configuration. This chapter reinforces the notion of interaction evolution in a ubiquitous system particularly over the long term: the concept of evolution used here is influenced by Dourish [57], MacLean [133] and Fickas [74]. Each of these authors identifies the ability to evolve, tailor and design a system by the user as a necessary feature for acceptance of ubiquitous or adaptive systems.

Interaction evolution is broadly defined here as *multiple related instances of interaction configuration (customisation or personalisation) over time that have a goal to change some aspect of the systems interaction behaviour.* For example, an elderly user might develop a visual impairment (e.g., cataracts) that requires a reduction in dependency on conventional visual displays.

Interaction configurations range from automatically generated rapid changes based on context to a process of modification driven by regular human reassessments of the system and its effectiveness.

The process of evolution is modelled as one or more potentially linked configurations, each

of which consists of the following stages, which occur iteratively:

- identification of opportunities for changing the system - an event occurs which indicates a change in the system or the environment which may allow (or require) a change in configuration to occur;

- reflection on alternative choices for change - the options for configuration can be analysed in order to determine which configuration option(s) should be used;

- decision-making - decisions on if the system should or should not be reconfigured (based on the previous reflection on the available options) and, if so, how it should be reconfigured;

- implementation - the chosen configuration is implemented.

Figure 4.1 shows this process as a spiral. The first configuration (1) shown by a solid line, shows a configuration that has gone through one and a half iterations while the second (2) indicated with a dotted line, shows another configuration that has only just been identified and the alternatives are under investigation. As shown in the figure it is possible to have multiple configuration processes under way at the same time at different stages of evolution. Each of these stages is now considered in turn.
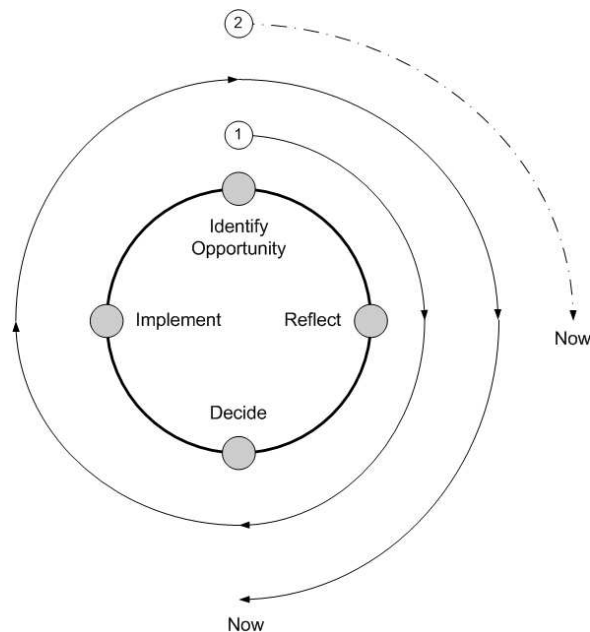
Figure 4.1: *Process of Interaction Evolution*

At first glance this is similar to the idea of autonomic computing [118] is an idea developed by IBM where autonomic elements manage themselves and integrate themselves into a

system based on high level orders or policies written by an administrator. However, autonomic computing is based upon the idea of a large number of self-regulating, self-managing components which operate independently within a system but which negotiate in order to cooperate with each other. Each autonomic component monitors itself in the form of a closed control loop similar to the one described in this section - continually checking to see if there any other autonomic components that it can use which are superior to the ones currently in use but the configuration loop used in autonomic configuration can only affect the configuration of a single autonomic component. Furthermore, the ability of the component to configure itself is limited by the goals that can be represented internally within the components.

These autonomic systems individually configure themselves while following policies which represent the aims or objectives of the administrator. Interaction between autonomic components is based on the idea of cooperative negotiation [24]. Autonomic components negotiate access to other components directly with these components. Simple forms of negotiation might be "first-come, first-served" where components providing a service will attempt to satisfy all requests until they run into resource limitations (in some cases it might be that only a single request can be satisfied). More complex forms of negotiation include bilateral negotiations over multiple attributes (price, service level, priority) and may involve counteroffers.

However, mechanisms for negotiating, enforcing, and reasoning about agreements within autonomic computing are lacking, as are methods for translating them into configurations [118].

Given the homecare focus of the MATCH project each step of this approach is illustrated by the use of a running reusable example taken from the homecare domain and validated by various stakeholders during focus groups and interviews with older users and health & social care professionals [36]. In this example, Fred and Shirley are an older couple with chronic conditions that could be ameliorated by appropriate use of ubiquitous homecare technology. In particular, Shirley has worsening arthritis and is no longer able to move around the house easily; she relies on Fred for tasks such as controlling the heating system, closing the curtains and for most household chores. Fred recently had a stroke. He is still physically fit but has become more and more forgetful since the stroke and requires continual reminders for when to take his medication. He is also hard of hearing.

They have a daughter Fiona who visits once a week and brings the shopping. A social care worker comes once a week and has offered them additional help with their shopping and household chores but Shirley and Fred are happy doing things for themselves for now.

4.2.1 Identify opportunity for change

For an adaptive system to evolve it is necessary to be able to identify opportunities for changing the devices and techniques the system uses to interact with the user. These opportunities are of many types, ranging from rapidly changing circumstances (e.g., ambient noise level) that need a rapid, probably automated change, to slowly emergent conditions that require rigorous (human) analysis and gradual resolution (e.g., deterioration of sight).

Identification of the opportunities for change within a system can include identifying the devices that are available, which are currently in use and which have been added and removed recently to the homecare system, as well as the available interaction methods or modality choices.

A candidate for interaction configuration is defined as as a combination of devices, interaction techniques, modalities used and supporting components required to instantiate a new configuration in order to satisfy the needs of an application task providing some functionality to the user (reminder notifications, streaming music, controlling devices etc).

Identification of opportunities for change can take many forms. They can be identified directly by Fred or Shirley, or another involved person, who can then take immediate action to rectify the problem. Alternatively they may gradually become aware of a deficiency in the current configuration before taking steps to address it.

In addition to identification of opportunities for change conducted by Fred and Shirley it is possible that as new devices, techniques or modalities are added to the system alternative, potentially better, devices can be substituted into current configurations in an attempt to improve them. This could be detected automatically by the system as it is running - perhaps using aggregate usage data of how other systems have configured this new component in the past to detect that it commonly replaces or is used in conjunction with existing components.

4.2.2 Reflect / judge alternatives

Once an opportunity for change has been identified, it is necessary to characterise the potential options for taking advantage of it. As with the opportunities themselves, the identification, characterisation and analysis of the options may be straightforward and automatable (e.g., presenting information to the user via the output devices currently nearest to them) or it may be complex, difficult to describe and evaluate (e.g., determining the alternatives for delivering a medical alert to a patient with progressive ocular deterioration)

perhaps needing the involvement of experts as well as decision-support tools. Since many systems (including homecare systems) are inherently multi-user it may be necessary to support collaboration between various stakeholders and assist in the description.

Once the options for change have been discovered, it is necessary to reason about the available options and determine their suitability. Some exemplar approaches of reasoning are discussed here, with reference to the example scenario presented in the introduction to this chapter, but also see Chapter 6 which provides a characterisation of different approaches to reflection and judgement of alternatives within the context of the model presented in the next chapter.

In an adaptive system, such as in a homecare environment, it is likely that users will have preferences for which devices or styles of interaction to use, but in a multi-user environment it is likely that Fred and Shirley will not have the same preferences all the time or in the same circumstances. Thus it is necessary to be able to incorporate multiple criteria into a decision making process on how to accomplish a task.

Fred and Shirley have different capabilities for interaction - Fred has difficulty hearing while Shirley has limited mobility. In this case, speech dialogue based interactions may make sense for Shirley as it eliminates problems with physically interacting with a homecare system, but may be an inappropriate choice for Fred. These conditions are likely to change over time and will need to be revisited periodically or when events force a change and this must be supported as an additional interaction within the system.

When a visitor is present, such as Fiona or the social care worker, this contextual change will affect the choice of method of delivering information to the couple. Reminders about medication or household chores may need to be suppressed while other people are present in the home - this problem is exacerbated when the information to be presented to the occupants is of a confidential or embarrassing nature. This requires that contextual information be included in the decision making process.

To support these, and other, types of decision that would need to be made, it is necessary to provide support for several different techniques for configuration which allow these decisions. It must be possible for users to be able to manually configure interaction - such that they are the ultimate arbitrator over a configuration and can have the maximum level of control at the expense of dynamic adaptability. It must be possible to include several analytical reasoning components which operate over the set of possible configurations. Examples of these might be location, preferences or contextual results such as ambient environmental factors which can be directly measured, analysed and decided upon. It must be possible to include techniques which interact with the user on an ongoing basis

to maintain relevance as opposed to a *fire and forget* configuration which would become less appropriate as conditions changed.

It may be possible to assess alternatives based on a record of their previous usage (e.g., identifying alternatives that have proved successful or otherwise in similar circumstances). This may be based on logging of user-system interactions or a record of special events of interest (indicators of satisfaction or dissatisfaction) about the current configuration. In addition to using this information to evaluate alternatives, it may be the basis of further evaluation, trying out new configurations on an experimental basis. Collaborative techniques, such as negotiated choices between interested parties or collaborative filtering [94], are clearly important in a multi-user home and it is necessary to support this ability to allow for conflicting sets of values to be combined to decide on the best configuration to use.

Techniques discussed in this section range from fully automatic techniques with no user interaction to techniques which involve ongoing interaction with the user as their primary concern and not all of these techniques are appropriate in every situation. The ability to allow for a range of manual and automatic reasoning techniques is a requirement for an effective decision on the correct configuration to use in making both short and long term changes.

4.2.3 Make decision

After reflection has taken place it is necessary to make a decision about whether a reconfiguration will take place; and if so what form it will take.

Both the decision itself and the resulting implementation of the configuration may be deferred until a later time - that is, the opportunity for configuration may be identified and recorded but the actual configuration does not take place until a later point in time. This may be required in situations where the user is currently busy and a change in modalities or interaction style would be a distraction to the task at hand. Alternatively, as in the visual deterioration case, the opportunity may be known (e.g., the rate of deterioration may be predictable) resulting in a plan for future reflection and decision making.

Decision-making, like reflection and analysis, may involve multiple agents and hence multiple criteria. Multiple stakeholders might be present in a homecare system (such as Fred, Shirley and any visitors to their home) and their independent criteria must be combined to make a choice of interaction, even if they are conflicting or contradictory, so that the determination of a solution can be made.

There are many different benefits and drawbacks resulting from the choice of combination

approach used to combine criteria and these are discussed in more detail in Section 6.5.4. Common issues arising from approaches to combining criteria from different sources are preventing dictatorship of one factor, maintaining pareto-efficiency and independence of irrelevant alternatives. However, ubiquitous systems are not necessarily limited by the same constraints that other voting or combination systems are - for example in some situations it may be the case that one criteria actually does matter more than others.

To cope with the issues presented by different types of system it should be possible that multiple such systems can be in use at the same time; both as combined evaluations incorporating multiple approaches as well as separate evaluations for different interaction tasks using different approaches.

4.2.4   Implement

Once a decision to make a change has been made it is necessary to transform an abstract decision into an actual change in the configuration of the system. The precise method of doing this will vary between systems but the required steps would be to identify the components that have been newly selected, identify components that should be discontinued and to arrange the transition that is necessary to switch from one to the other.

Implementation is the ultimate proving ground for a configuration choice. Although it is possible to guess or reason about the suitability of a configuration, it is only by implementing it that it can be discovered if it is truly appropriate for the circumstances.

4.2.5   Iterate

This entire process of handling change is iterative and ongoing to support evolution of interaction. People do not necessarily know in advance which interaction techniques and devices will and will not work in different circumstances and may need to experiment before deciding. This implies that each iteration would include an evaluation phase as part of identification of opportunities for change to determine if the new configuration meets the needs of the users better than it did previously. The users would typically have to be involved in this step to make this judgement.

Over time, new criteria will emerge that will need to be reasoned about in order to choose the best candidate for configuration. Examples of these might include new people or devices moving into the home or a change in the criteria that should be applied. To do this it must be

possible to add new techniques or change the techniques in use within the home by allowing additions or removals from the active models at runtime.

The concept of ongoing re-evaluation, discussed previously, requires a process of evolution to improve the situation over time and through changing circumstances. To accommodate this it must be possible to decide when it is appropriate to perform these evolutionary steps. It may be desirable to change the active configuration as soon as a new device or context change occurs, but in some circumstances it may be desirable to limit the number of changes that take place or to cause them to occur at a fixed time or after a certain other event has taken place.

4.3 Overview

This chapter has presented a process which encompasses the primary concepts within interaction evolution as a four stage iterative process. This is a continual approach and is applied on an ongoing basis, this allows for an iterative decision making process to support evolution of interaction. This design for system support builds upon the ideas for evolution as multiple related instances of personalization or customization by allowing for multiple instances of reflection and judging of alternatives which are capable of individually reasoning over the candidates for evolution which are then combined to allow for decision making to take place.

The research questions presented in Section 1.1 ask how a system can be modelled to answer the questions below:

- What is the system currently doing?

- What can it do?

- How can it be changed?

The process that has been presented is regarded as having the minimum necessary steps that are capable of addressing each of these questions.

Identification of opportunities for change - Identification of opportunities for change is required to address the question *How can it be changed?* as the enumeration of choices of how a system can change depends upon being able to identify the components that can be changed within a configuration at the appropriate time that they can be changed.

Reflect / judge alternatives - Without the ability to reflect upon the available opportunities for change it is not possible to ask *What can it do?* which implies an ability to differentiate

between the available options to reason about the available opportunities in a way which allows them to be chosen from.

Make decision - The need for an ability to make a decision on what the system should do is derived from the questions *What can it do?* and *How can it be changed?*. These questions both infer that one or more of the things that it can do will be selected before asking how the configuration can be changed to this selection.

Implement - Once such a decision is made it is necessary to be able to actually implement it. Without being able to do so the question to determine *What the system is currently doing?* is not possible as the current configuration can not exist without it being implemented at some previous point in time.

Iterate - Iteration is important as each of these questions can be asked repeatedly. Once a new configuration has been made it is still possible to again ask these questions of the new system configuration which requires an acknowledgement of the iteration within the process.

As should be evident from this chapter, the process is best viewed as a collaborative activity involving multiple human stakeholders interacting with the system itself (the target of change) and potential computer-based support tools in order to make decisions. For that reason, an approach that attempts to link these aspects via a common model is necessary. Such a model is presented in the next chapter.

5

Configuration Model

The previous section discussed the process of interaction evolution for configuration of interactive system through a series of steps starting from identification of opportunity for change and working through reflection, decision making and implementation of the configuration options in an iterative process.

This section introduces a model based approach to configuration of interactive systems based on the concept of *evaluation functions* which satisfies the requirements of this process while providing a number of compelling features. This approach represents each of the techniques that can be used for configuration within a unified model. This approach allows designers to provide many configuration techniques in parallel or in combination that are

potentially modifiable at run-time and capable of being driven by user interaction. To illustrate this approach the application scenario presented in the previous chapter will be used throughout this chapter.

5.1 Application Context

This work has been carried out as part of the MATCH project and for that reason, the approach is illustrated by the continued use of a running example taken from this domain, and continued from the previous chapter. Recall that Fred and Shirley are an older couple with chronic conditions that could be ameliorated by appropriate use of ubiquitous home care technology. In particular, Shirley suffers from worsening arthritis and is no longer able to move around the house easily; relying on Fred for tasks such as controlling the heating system, closing the curtains and for most household chores. Fred is hard of hearing and recently had a stroke and, although still physically fit, has become more and more forgetful since the stroke, requiring continual reminders to take his medication.

5.2 A Unified Model of Configuration

The model presented here is designed around the concept of *evaluation functions* that are responsible for both identifying opportunities for change as well as reflection on the alternatives available to make a change.

The concept of a *configuration possibility* (hereafter, *possibility*) is introduced which is an *encapsulated solution (consisting of interaction elements, techniques and devices) that can offer interaction between a system task and a user*. In terms of the model presented here the possibility may be composed of functional elements which are abstract concepts, software or hardware, but in a software implementation these must be realisable as software or must be able to be interacted with using some software component. Accordingly, a possibility includes any software elements needed to perform data transformations related to the interaction as well as references to the elements that will be responsible for rendering the interaction via physical devices.

Possibility. *A possibility p is defined as an ordered n-tuple consisting of elements from the set of elements E (consisting of available interaction elements, techniques and devices) that can be used with an application task t from the set of all application tasks T. Formally,* $p = (t, e_1, e_2, e_3 \ldots e_n | e \in E, t \in T)$
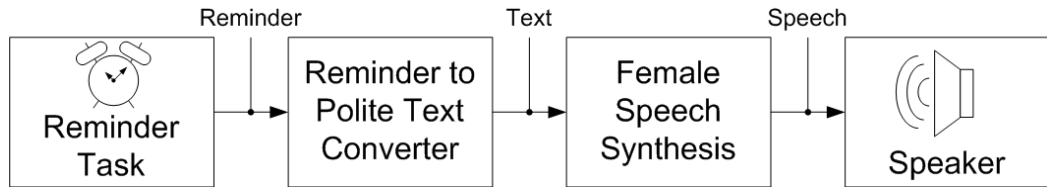
Figure 5.1: *A typical configuration possibility*

Consider a medication reminder for Fred; one of the possibilities, as shown in Figure 5.1, might be to deliver the reminder via a speech synthesis system. The possibility would include the software representing the physical device (the speaker), the speech synthesis system (responsible for converting text to speech) and the software that converts a medication reminder into the appropriate textual alert.

A key concept in this work is the notion that it is possible to model an interactive system as a directed graph of available elements - from which the available possibilities can be derived. This graph can be constructed using a service discovery system that models relationships between available elements. A directed graph is typically defined on a given set of vertices *V* (in this case members of the set of elements) and edges *E*.

Possibility Graph. *A possibility graph G is defined as a pair (V,E) where V is a set of vertices $(V \subseteq C)$ and E is a set of edges between the vertices $E \subseteq \{(u,v) \,|u, \ v \ \in V\}$*

By identifying interaction elements, it is possible to traverse the graph with the goal of constructing a set of possibilities that can be used with the application task. Sections 7.3.2 and 7.3.2 discuss in detail how to build and traverse such a graph later in this thesis.

Traversal. *A traversal is defined as a function (traverse) which operates over a graph (G) parameterised with an application task (t) and returns a set of possibilities (P) such that $P = \{p_1, p_2, p_3 \ldots p_n\}$ where $P \leftarrow traverse_{(t)}(G)$*

Figure 5.2 shows a typical, albeit simple, graph that may be constructed from the data in a service discovery system. In this graph different possibilities can be deduced (such as the speaker using polite text and a female voice); shown is a speaker that requires the choice of two of the intermediate elements as well as a GUI that does not require intermediate elements. By starting from the reminder task as the root node, a graph traversal can be performed to determine each possibility in the graph.

Once the graph has been built and traversed to create a set of possibilities, it is possible to analyse the appropriateness of each possibility. To do this each possibility is evaluated by
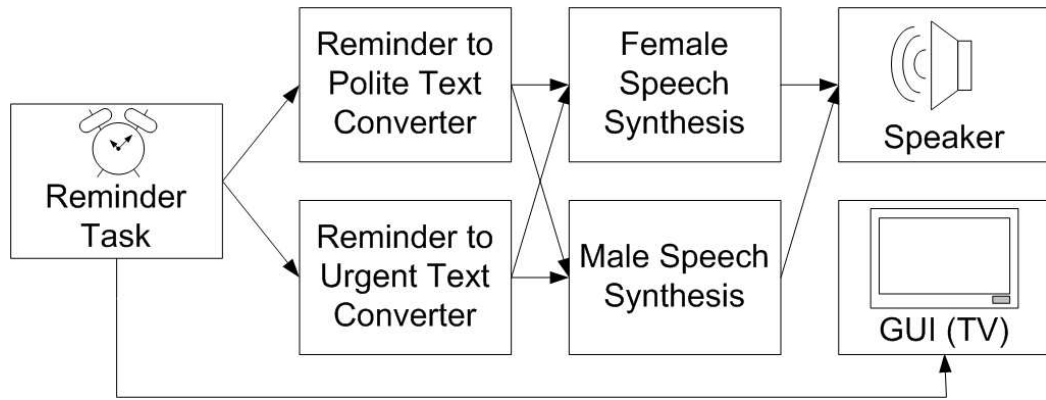
Figure 5.2: *A typical graph*

using one, or many, *evaluation functions*.

The purpose of an evaluation function is to rank, filter or otherwise analyse these possibilities to reduce them to a set of selected possibilities which represent a configuration decision that has been made. Evaluation functions can have a many-to-many relationship with task assignments; there may be many evaluation functions used to review the possibilities for the medication reminder task while a single evaluation function may be used simultaneously for many tasks. Here approval and ranking evaluation functions are presented; which allow/disallow possibilities and assign scores to possibilities respectively.

Approval Evaluation Function. *An approval evaluation function (φ) is a restriction (σ) over the set P producing the set P' (a selection of appropriate possibilities from the set of available possibilities). Formally, $P' \leftarrow \sigma_\varphi(P)$*

Ranking Evaluation Function. *A ranking evaluation function the result (P') of applying a metric function (M) to score the possibility to each entity of P. Therefore, $P' \leftarrow \{(p_1, m_1), (p_2, m_2)...(p_n, m_n)|p \in P, m \leftarrow M(p)\}$*

Figure 5.3 shows one possible result from the application of two evaluation functions (a ranking and an approval function) to some of the possibilities that have been generated in the previous step. The Usage History Ranking is an example of an evaluation function which uses the recommender approach to rank possibilities while the Doctor's approval function allows or disallows possibilities; here the Male Speech synthesis is disallowed as it sounds too similar to Fred and can confuse Shirley.

To allow multiple evaluation functions to be used with a single task it is possible to use evaluation functions to combine results via function compositions (in effect a *meta-evaluation function*). This allows the results of multiple approaches (implemented as
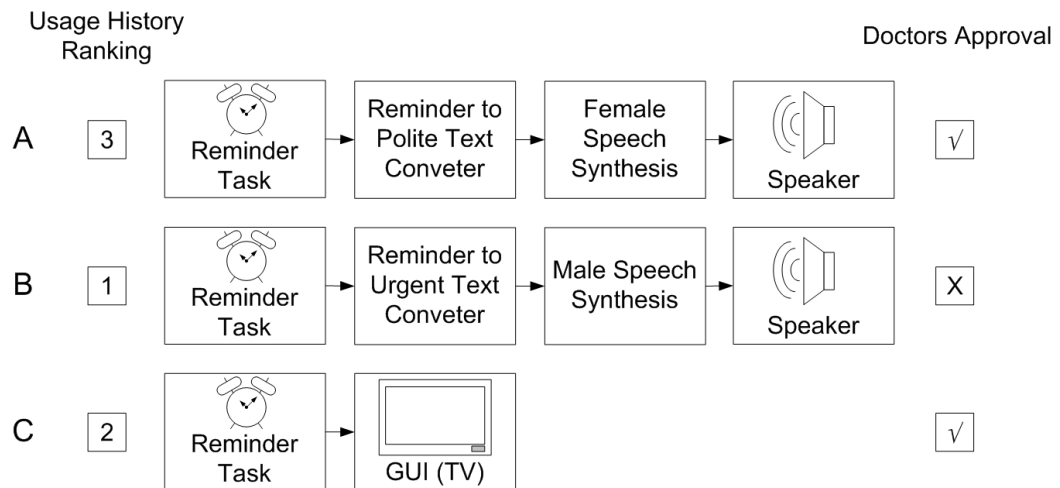
Figure 5.3: *Example results from the application of a ranking evaluation function and an approval evaluation function.*

evaluation functions) to be combined together into a single function that can be mapped onto the task.

Meta-Evaluation Function. *A meta-evaluation function is any function (F) which produces a new set of possibilities (P′) from combination of the results of two or more existing sets of possibilities ($P_1, P_2 \ldots P_n$). Specifically, $P' \leftarrow F(P_1, P_2 \ldots P_n)$*

This approach would allow, for example, the selection of an interaction technique for the notification task to be based on a combination of context sensitive, manual and/or automatic reasoning. A typical example of this might be that the users' preferences are weighted against the results of a collaborative filtering system receiving input from multiple users, based on the success of similar tasks.

Figure 5.4 shows one possible method by which the previous two evaluation functions (one ranking and one approval) might be combined together to determine which possibility to use from the three available possibilities shown in Figure 5.3.

As the Usage History Analysis evaluation function is implemented as a ranking function it scores each of the possibilities and the results are combined with the results of the doctor's approval evaluation function by a meta-function which removes any of the ranked possibilities which were not also approved. A final meta-function selects the possibility with the lowest numerical rank (corresponding to its position in the rankings). Possibility 'C' is the possibility with the lowest numerical rank that had also been approved and was therefore selected.
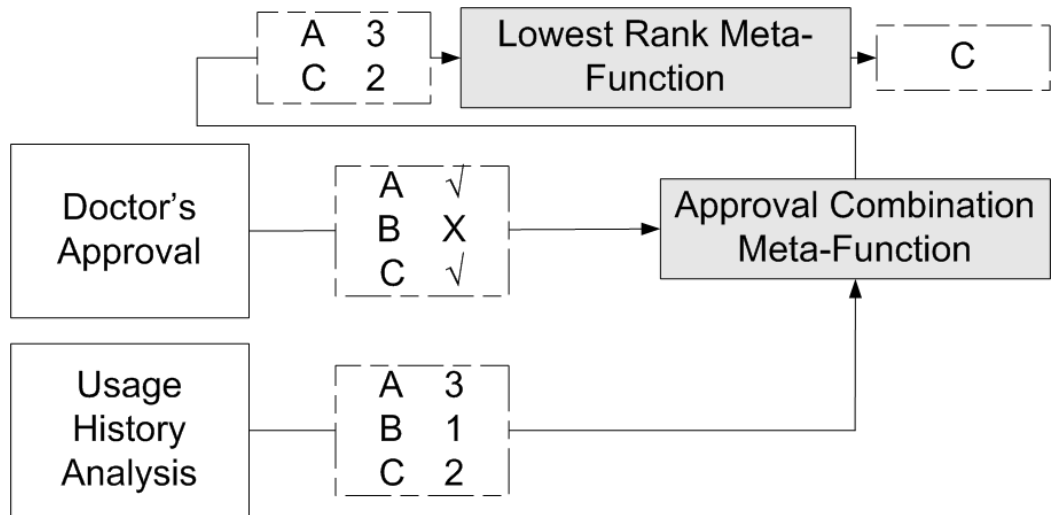
Figure 5.4: *Example results from the combination of two evaluation functions.*
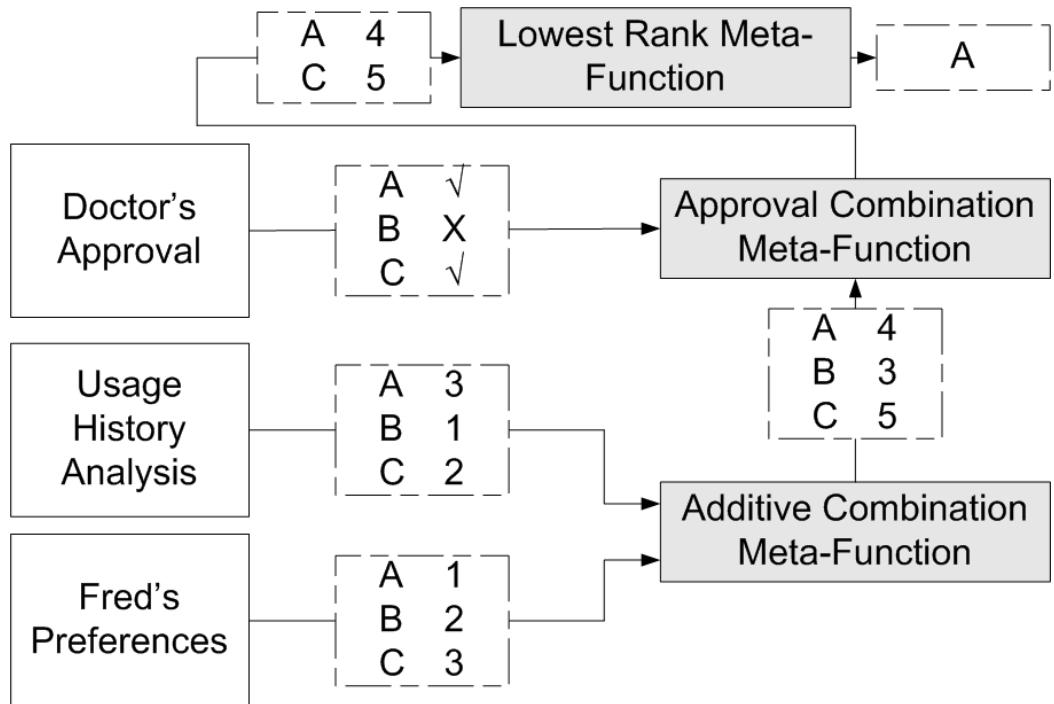
Figure 5.5: *Example results from the combination of three evaluation functions.*

Functions can be replaced or changed at will to provide different results; Figure 5.5 shows how introducing a third evaluation function into the tree might affect the results; the individually ranked results of both ranking functions (Usage History & Fred's Preferences) are first combined together using a meta-function which combines the separate numerical

rankings, e.g. via weighted combination, to create new ranking before that is then further combined with the doctor's approval evaluation function and evaluation proceeds as previously. In this case the selected possibility is now 'A' as it has the lowest numerical rank that has also been approved.

The choice of meta-function to combine the results of the two ranking functions could have instead been multiplicative in nature which would again have had a different result.

One feature of this is that the system has native support for combining multiple criteria within a single evaluation process. Each criteria in the task can have their own evaluation function(s) modelled after their views or requirements - the results of which can then be combined within the same framework. This allows the natural specification of how criteria can be combined by changing the meta-evaluation function being used to combine the results.

The eventual result of an evaluation function (or tree of evaluation functions) should be the set of possibilities to use for interaction, as shown in Figures 5.4 and 5.5. In this case, a single technique has been selected, although functions might enable multiple concurrent techniques to be used.

Evaluation functions are a flexible method of reasoning about the available possibilities and can be applied at different levels of granularity; some evaluation functions may consider an entire possibility while others may only operate over selected portions of a possibility. For example, an evaluation function may only consider the choice of physical output device in its reasoning. Evaluation functions may utilise external sources of data such as context or usage history and can be parametrisable such that a single evaluation function may be reused in multiple situations (such as gathering of user preferences from multiple stakeholders) or even called recursively.

It should be clear that, although the example used here is based on a single output of a single application task, this model is generalisable to inputs as well as outputs, multiple instances of the same application task and to multiple disparate tasks.

5.3   Further Examples

In this section a number of further examples will be presented to introduce some of the features of this approach at a very high level and to demonstrate some practical applications that it allows. These examples are deliberately very simple in order to highlight the benefits of the approach from a user level. The approaches, methods and mechanisms are explored

in more detail in the next chapter. The examples listed here are somewhat arbitrary and are chosen to express different concepts rather than as a realistic example of how Fred and Shirley's thought processes might work.

Example 1 - Analytical Functions, Multiple Resolutions

Imagine that it is necessary to inform both Shirley's doctor and Fred of Shirley's condition on an ongoing basis. There would exist a task (or similar computational agent) which collates the information necessary to do this on an ongoing basis. It is then necessary to decide how to inform each person of Shirley's condition. In this example (and later examples) assume that the set of available possibilities includes:

- HTTP post submission to a shared monitoring screen at the doctors surgery;

- SMS to the doctor's phone (provided as a backup to the monitoring screen);

- a television in the living room;

- a loudspeaker which is audible throughout the house;

- a monitoring application on Fred's mobile phone.

Acceptable choices may change over time. Imagine that it is possible, however unlikely, to identify the approach to use to contact each person based on a simple indicator such as time of day based on the advise of Shirley's doctor who was responsible for defining the desired behaviour.

It would then be possible to define a function that, given the current value of the indicator and a set of available possibilities, was able to identify the two approaches from the above list to use at any given time. Such a function can then be represented as an evaluation function which selects the two best possibilities representing the possibilities which should be used for contacting the two respective persons.

The key ideas represented by this example are that it is possible to encapsulate domain knowledge (although unrealistically done so in the example) within an evaluation function and that evaluation functions can return multiple approved possibilities.

Example 2 - Manual Configuration

Imagine that the provided evaluation function from the previous example would frequently select the HTTP post submission as well as the audible loudspeaker to deliver the information. In short, the evaluation function did not meet the requirements of the situation; the frequent loudspeaker announcements are annoying to Shirley and difficult to hear for Fred.

To resolve this, Fred and Shirley decide to manually specify the devices to be used. This

can be performed by creating an evaluation function which has some mechanism, discussed later, for interacting with the user to select the current choices. The evaluation function can then store that choice and select the chosen possibilities when queried.

Note that it is not necessary to change any of the underlying infrastructure to enact a change from a completely automated approach to a completely manual approach - only the choice of evaluation function associated with the task is changed. Additionally it is important to note that the evaluation functions can have side effects (viz. interaction with a user).

In this scenario Shirley selects the HTTP based surgery monitor and manually updates the evaluation to select either Fred's phone or the television in the living room depending on whether or not Fred is home.

Example 3 - Simple Preferences

Eventually, despite the additional control that manual configuration provides, Shirley tires of manually changing the device between Fred's phone and the television and decides that she wants to encode her preference between the possibilities.

Fred defines his preferences (Phone > TV > Loudspeaker) and chooses an evaluation function which will respect these. Choosing the highest ranked possibility from his list of preferences at any one time. i.e. if the phone is available then the phone possibility will be used, otherwise the television and finally the loudspeaker.

These preferences could be stored and maintained in a variety of ways. For the purposes of this example it is assumed that there is an external preferences store that is queried in order to obtain this preferences data and that this is maintained separately such the notion of an evaluation function accessing a source of external data is introduced.

Fred begins to turn his phone off when he's in the house to exploit this preference feature so that it is marked as unavailable and is therefore not provided in the list of available evaluation functions and thus cannot be selected by the preferences evaluation function. This causes his second preference, the television, to be used instead.

Example 4 - Combining Evaluation Functions

This usage of the preference function is only a partial solution to the problem as it only supplies Shirley's condition to Fred and not the doctor. The preferences evaluation function could be arranged such that the preferences were: Phone > HTTP Post > TV > Loudspeaker > SMS and the top two possibilities were chosen but this is a messy solution to the problem. If the Phone and the HTTP Post were both unavailable then messages would be delivered to the TV and the Loudspeaker and the doctor would not receive a message.

A cleaner approach would be to separate the preferences and combine them. One

preferences evaluation function would be configured with Freds preferences (Phone > TV > Loudspeaker) while another would be configured with the doctors preferences (HTTP Post > SMS).

Each of these preferences functions would select the most preferred possibility from the available possibilities at the time - but since the selection is performed independently there is one preference for Fred and one for the doctor.

These selections are then provided as input to a third evaluation function which is responsible for combining the results - in this case it only needs to return both of these two possibilities as its result.

Example 5 - Context Sensitivity

In the previous two examples, Fred has had to turn his phone off when he enters the house to cause the preference based system to switch to using the television. This situation is not ideal since Fred may receive phone calls while his phone is turned off.

To address this problem, it is decided that Fred's preference evaluation function should be replaced with a context sensitive evaluation function to control the configuration based on Fred's behaviour. Here the appropriate contextually sensitive evaluation function would detect if Fred is at home or not and return the appropriate possibility. Other contextual evaluation functions which might be used by Fred and Shirley are monitoring of light levels to determine which rooms are in use to only use devices available in those rooms, or monitoring ambient sound levels to adjust the volume of audio alerts or to determine if they are appropriate at all.

A generic version of these contextually sensitive functions might be to create a context sensitive function which acts as a switch between the results of two other evaluation functions (that is to say; between your preferences in one situation vs. your preferences in another situation). Trees of evaluation functions can be created where each of the inner branch nodes within the tree are responsible for an aspect of context sensitivity - allowing contextual data to be combined together.

It is possible that the actual data being monitored could be contextual, such that if Shirley has not moved for an extended period of time then the choice of interaction technique might change (i.e. to send an SMS to the doctor's phone) rather than using the passive monitoring provided by the surgery.

5.4 Overview

In this chapter a model is presented composed of a number of key components. These are: (i) possibilities which encapsulate a particular implementable configuration, (ii) a graph representation of elements within possibilities, (iii) an ability to traverse this graph and obtain sets or lists of possibilities as a result, (iv) evaluation functions which can reason over these sets of possibilities, and (v) an evaluation function tree structure which allows for combination of disparate approaches. The execution of this model results in a determination of which possibilities should be implemented.

These features fit naturally into the process presented in Chapter 4. To recap, the key features of that model were;

- identification of opportunities for change

- reflection on alternatives

- decision-making

- implementation

- iteration

The model presented in this chapter supports each stage of the process as follows:

Identify opportunities for change: Identification of opportunities for change can occur in a number of ways within the model. The most obvious of which is reconfiguration of the graph (or the source of the graph) by addition, removal or modification of any of its entities. However, this is only the most coarse form of identification available. Recall from the examples in this chapter that evaluation functions can respond to external events or incorporate external conditions into an evaluation. Chapter 6 explores how evaluation functions can be used to identify opportunities for change from (amongst other things) context, changes in environment and interaction with stakeholders.

Reflect / judge alternatives: The model proposed in this chapter is specifically designed to allow for extensive reflection on alternatives and allow this reflection to be orchestrated by computational methods, by human interaction or by some combination of these. Specifically, evaluation functions implement a *unit* of reflection or analysis which can be performed.

Make decision: Likewise, the decision making process is explicitly modelled by the ability to combine evaluation functions (units of reflection) together to arrive at a decision on which configurations should be implemented. As this decision process is a combination of

evaluation functions it can incorporate any number of varied approaches, combinatorial or human centred, to arrive at the final decision.

Implement: Implementation of the selected configuration is accomplished by the explicit representation of a unit of configuration for a task in the form of possibilities. Possibilities contain all the information necessary for the system to implement a particular configuration.

Iterate / repeat: Finally, the process allows these steps to take place in an iterative and ongoing process. The model presented here allows for a large number of temporal properties which will be discussed in detail in Section 6.6.

The next chapter provides a characterisation of the configuration space which this model fits into and discusses how this model can implement a variety of different techniques from within this configuration space.

6

Characterising the Configuration Evaluation Space

The previous chapter presented a model that is capable of supporting a range of different evaluation techniques. This chapter discusses, firstly, the characteristics of such an approach and, secondly, presents a categorisation of the range of evaluation functions that can be implemented within this model.

The approach that is described in the previous chapter relies on a number of key features in an underlying system. Before describing how additional features can be used within the approach and their characteristics, it is necessary to first formalise the characteristics of the model described in the previous chapter.

Characteristics of an adaptive system by a particular feature will be highlighted throughout this chapter. It should be noted that the majority of characteristics identified here are only applicable for specific features or functionality within the model.

6.1 Assumptions

This chapter discusses the configuration evaluation space of adaptive systems and in particular, how this design space maps back onto the notion of evaluation functions and the model presented in the previous chapter. Firstly, assumptions for the purposes of discussion are identified which, although not actually necessary for the model presented, are useful for the purposes of taking into account preexisting methods and techniques.

It is assumed that any system implement the model will conform to current best practise with respect to software architecture and engineering; including a modern object oriented conceptual framework is used. It is possible to implement these ideas without using object-oriented technology but this assumption is made in order to allow for clearer discussion of the following sections.

Characteristic 1. *Systems that implement the model use an Object Oriented conceptual framework.*

As the approach described in the previous chapter is modular, it must be possible to describe the constituent parts of the system in terms of modules or components. As such the approach is dependent on being built upon a component based model for describing the available facilities within the system. This model may be implemented as objects within a programming language, services (such as SOAP XML message description) [71] or as formal descriptions of the mechanisms required to interact with a service which offers this functionality [138].

Characteristic 2. *Features are described in terms of a component-based architecture.*

A crucial feature within ubiquitous systems is that the selection of available components in a system is adjustable at run time to allow new subsystems or features to be added without the need to stop or reinstall the system. That is, new code can be automatically or dynamically loaded into the running system and be detected by the system to allow new features to be added (or old features to be updated) without the need to stop or restart the system. It should be obvious that while the approach described in the previous chapter has this capability it can only be implemented when the underlying component model makes this possible.

Characteristic 3. *Components can be added or removed at runtime.*

There are a number of component models that could be used to accomplish this, some of which, such as the Component Object Model (COM) [191], the Distributed Component

Object Model (DCOM) [29] and Java RMI [59] are designed to deal with challenges of process intercommunication but can be used to swap processes or components at run time. Other approaches such as the Common Object Request Broker Architecture (CORBA) [206] or OSGi (previously known as the Open Services Gateway initiative) [138] are capable of loading components within the same address space as an existing application.

A further characteristic of the component model is that it provides a known or derivable type system that allows determination of component compatibility in terms of compatible interfaces or data types. Briefly, it must be possible to determine that two components can be used together.

Characteristic 4. *It is possible to determine if two components can be used together.*

There exist two main approaches to determining compatibility between components. The first approach is to establish an agreed upon interface that a component is willing to communicate with. A task component could, for example, define that it is capable of interacting with other components which implement a specific interface or one of a collection of interfaces that it understands. Interfaces must have an explicit restriction on the type of data that they accept as part of the signature of their methods.

The alternative approach is that the dependency on interfaces is eliminated and instead the data types a component can use are defined. In this approach, a component which consumes a type of data is regarded as being compatible with any other component which generates it.

In addition to being able to determine compatibility between components it is necessary to be able to allow the evaluating component to bind the target components together to allow them to communicate. *First party binding* occurs when an application explicitly finds and defines its dependencies using import statements or a similar declaration and uses these components directly but which assumes that the information necessary is compiled into the application or provided as an instantiation parameter [43]. *Third party bindings* are performed by an external third party (the manager) and the objects being bound do not take an active part in the binding [43] which allows other components in the system to connect it to other components for it. Third party binding is considered a requirement for efficient or transparent reconfiguration [82].

Characteristic 5. *A component must be able to be connected to others via third party binding so that its connection to other components can be controlled by a third component.*

Third party binding can be layered on top of a first party binding system, either:

- by including specific control methods into components which allow them to have their binding controlled by third party components

- by including a control channel which allows their binding behaviour to be controlled via messaging

- or by utilising object oriented patterns; such as proxies which behave as third party bound components but access underlying services directly via first party binding.

In order to locate components themselves, as well as determine their compatibility, the information about available components must somehow be made available to a third party component.

Characteristic 6. *A discovery system is required which can be used to locate components and identify their interfaces or data requirements.*

A service discovery system can be implemented in a variety of different methods. The UPnP protocol [123] for example uses a broadcast approach to advertise available components which is known as the Simple Service Discovery Protocol (SSDP) [123] which allows components to seek out other UPnP devices on the same local network segment. Another approach is to use a central registry of available components such as the Universal Description Discovery and Integration (UDDI) repository [160] which is available for use by Web Services based applications.

The characteristics of the basic approach were presented in Chapter 5 and will be expanded upon in this section by providing a summary of different types of evaluation function that could exist within this approach, detailing their function and form as well as specifying any additional features that would be required of an underlying system in order to implement them. This chapter will serve to illustrate the extensible nature of the approach.

6.2   Configuration Evaluation space

There are a large number of criteria that could be used to decide which interaction technique or device should be used - to demonstrate the expressiveness of this model, some of the relevant factors will be discussed within the context of the framework described in Section 2.4.1 by Thevenin and Coutaz [217]. This framework was discussed in Section 2.4 and is extended and revised in the remainder of this chapter.

Briefly recapping, Thevenin and Coutaz's design space, shown in Figure 2.11, is based on four orthogonal axes: Target, Means, Time and Actor.

- Target - The target axis refers to the entities (users, system, environment) which the adaptation should accommodate. This axis includes situations in which the system adapts to the users of the system, the environment or to the system characteristics. This includes context sensitive adaptation or where the selection of appropriate interaction technique depends on device characteristics.

- Means - This refers to software components involved in adaptation. This includes mechanisms such as policies or mechanisms for combining criteria or other techniques which are involved in the adaptation decision making process.

- Time - The temporal axis refers to different approaches to the ordering or timing of adaptation. Thevenin and Coutaz identify two broad approaches of static and dynamic approaches to adaptation but a number of additional alternatives and refinements to this categorisation are presented in this chapter which also lie along this axis.

- Actor - This axis deals with the agent responsible for triggering the adaptation. These include multiple human stakeholders and machine actors. It should be noted that this represents a different logical function from the target axis; adaptation may occur triggered by one entity (the actor) but intended to adapt to another (the target).

The Time and Actor axes are left open and not refined in detail by Thevenin and Coutaz. In addition to these four axes, I propose the addition of an orthogonal axis to represent the Source of factors which affect adaptation or configuration.

- Source - This axis represents the fact that there are a wide variety of sources of data, both internal and external, that can be used during the adaptation/configuration process. These data sources may be complex, they may be entirely separate from the application and there may be many of them.

This additional axis results in a five dimensional nominal unordered configuration space, shown in Figure 6.1. My refined axes includes the original four axes in addition to one additional axis. This results in three axes which were either absent (source) or not discussed (time, actor) in the original paper.

Readers are reminded that the complete configuration space is, in reality, more complex than can readily be expressed within a single document - in particular it is important to realise that each of these axes have a considerable amount of substructure that may or may not be fixed.

One example of this is the *User* entity within the Target axis. A group of users may be categorised in a virtually limitless number of ways - such as hierarchically by position in a company or organisation, by role, by gender, class or ownership of devices or components.
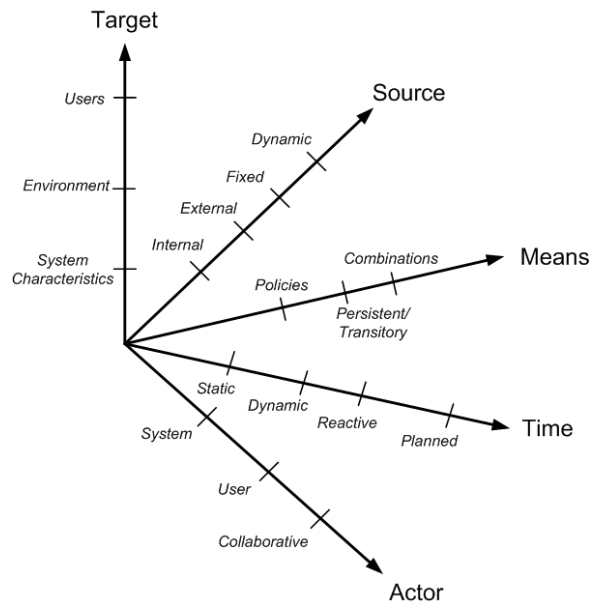
Figure 6.1: *Revised Adaptation design space*

Although this chapter explores the structure and members of each axis it does not impose a specific fixed substructure and these should be regarded as being fluid and subject to change.

Given this, this categorisation is primarily interested in discovering the important categories of factors which affect configuration which are represented by these axes and furthermore finding exemplars within each factor which elucidate the properties of the factor and show how they can be managed and dealt with within the model concerned by this thesis.

The remainder of this chapter will discuss each of these axes in turn and illustrate the discussion with example configuration techniques which can be placed along these axes. For each of these techniques it will be shown how they could be implemented as evaluation functions within the model presented in the previous chapter.

6.3 Target

In order to discuss the types of evaluation function that may exist, it is necessary to first explore the range of attributes that an evaluation function can reason about during the evaluation process. The Target of an adaptation is typically the most discussed in existing literature in regards to adaptation. The three canonical targets for adaptation, as described

by Thevenin and Coutaz, are (i) the system physical characteristics, (ii) the environment and (iii) the user.

The system physical characteristics refer to choices made due to limitations in the systems ability to offer a particular adaptation for purely technical reasons. A prime example of this would be restriction of options based on device availability; if no speakers are available, or the speakers are currently in use for another higher priority function, then audio will not be available. This may include situations where some options, which are theoretically available, are not practically available due to limited resources or excessive burden on a shared resource; such as insufficient CPU available for a speech synthesiser on a shared system.

The environment, within which the configuration resides, provides a rich source of targets to adapt to. The environment target represents all the different physical factors evident in the real world which may affect the choice of configuration. The user target is similar to, but distinct from, the environmental target and represents choices based on the properties of the user(s). There may be zero, one, or more users who may be singularly and jointly targets for configuration.

The environmental and user targets are often discussed jointly in existing literature; a summary of the environmental and user targets, and properties of these targets, is provided by Schmidt [200] who makes the central argument that context is more than location.
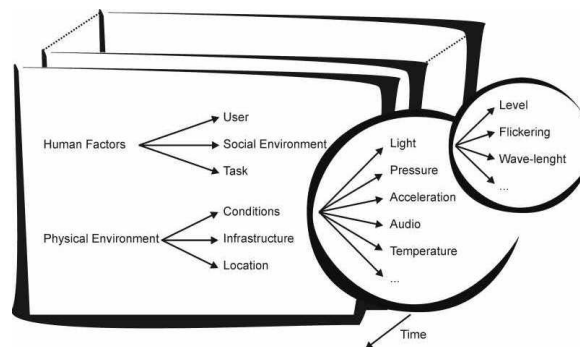
Figure 6.2: *Target Properties - Context Feature Space from [200]*

Schmidt describes each of the two targets as being composed of three categories which can be further subdivided into more specific features. The categories used for the Environmental target are location (either relative, absolute or co-location), infrastructure (available resources) and physical conditions (noise, light, etc).

Schmidt goes on to describe the categories of the user target as being information on the

86

user (habits, emotional state, conditions), their social environment (co-location of others, social networking and group dynamics) and the users tasks (this and other activities they are involved in).

It should be noted that Schmidt regards the system's physical characteristics as being a subcategory of the environment target rather than at the same level of importance as the User and Environment as Thevenin and Coutaz do. In addition, location is described as a property of the environment rather than of the user whereas there are convincing arguments that this should be inverted; particularly in the case of multiple users and therefore multiple locations which could be used.

Pragmatically, the precise categorisation of the target axis is not the focus of this thesis so no argument as to details of which targets are more important than others or how each target should be categorised and sub-categorised is made here. Rather, the aim is to highlight the existence of a variety of valid targets and to elucidate the vast range of properties which can be used to inform the configuration process. The next section describes various approaches which can be used to obtain information about, or from, these targets.

6.4   Source

In this section, the general techniques for retrieval of information about the targets, discussed in the previous section, is described. There are two main approaches which can be used; these are (i) reasoning directly upon elements in a possibility and (ii) reasoning about the possibility using some form of external data about the possibility. A number of different types of evaluation functions using these approaches are discussed in this section.

6.4.1   Possibility Attributes

The most simple type of reasoning is to choose possibilities based entirely on the properties of the possibility - for example the choice of an intermediate element within a possibility might restrict the maximum sample rate of an input source and as such this would be ranked lower than possibilities which did not include this element.

In addition to observations made of the possibility as a whole, an evaluation function may make decisions based on attributes of a possibility or an element within the possibility; for example, the name of an output component may be used to implement a function that approves only components with a particular pre-specified name or identifier or created by

a particular manufacturer). This reasoning can be applied to the intermediate components within a possibility; such as favouring pleasant female speech synthesis over an alternative more urgent male voice.
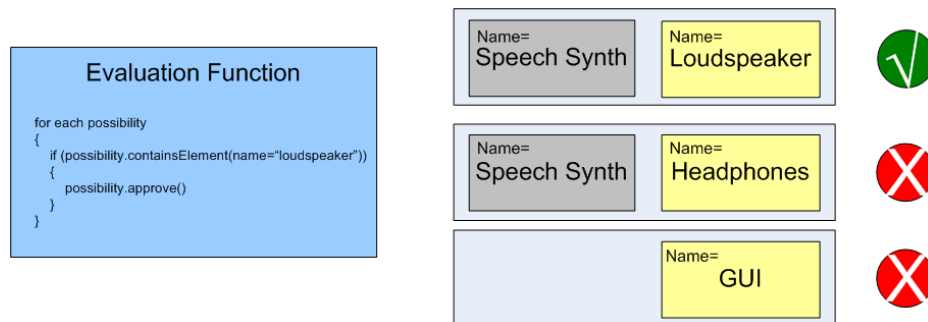
Figure 6.3: *Internal attributes of a possibility*

This approach requires that possibilities can be assigned properties and that the evaluation function is able to inspect possibilities to retrieve the appropriate information directly from the possibility and furthermore identify the properties that it is looking for (such as possibilities containing a property identifying it as a particular named component). This is the basis of Characteristic 7.

Characteristic 7. *Possibilities, and elements within a possibility, can be assigned metadata which can be accessed by evaluation functions.*

Some of these possibilities may have attributes which are always available and veritably correct (such as the length of the possibility) while others (such as the meaning of a name applied to the possibility) may require additional information to be added to the possibility in order for it to be accessible and its dependability may vary depending on factors outwith the system itself. Properties of possibilities may be both categorical (such as data types used or presence of particular properties) or empirical (such as numbers of elements within the possibility). Alternatively possibilities may be inspected using a population approach [34] which considers each possibility as a member of a large population of other possibilities - some of which it will be more similar to than others - where behaviourally (if not structurally) similar possibilities may then cluster together after usage successive iterations by a large population of users.

6.4.2 External data

Rather than restricting the criteria that can be used to make decisions about possibilities to only those which can be directly derived from the possibility, external sources of data can be introduced to provide a richer source for information about targets. This involves making decisions based on information that is not directly supplied in the possibilities themselves. In this chapter references to *external* data refer to that data which is external to the evaluation function - it may not necessarily be data which is external to the entire system, as it may be contained or controlled within another component of the system.

In contrast to data that is solely internal to the evaluation function, data that is obtained externally to the evaluation function may take a number of different forms (access to a shared database, ontology or other resource/information oracle that provides a service to the evaluation function) - each of which can have its own set of benefits or restrictions. This source of data may be computational, as in databases, but may take the form of user interaction (either brokered by the evaluation function or by some external arbitrator) where the information source is actually direct interaction with the user themselves.
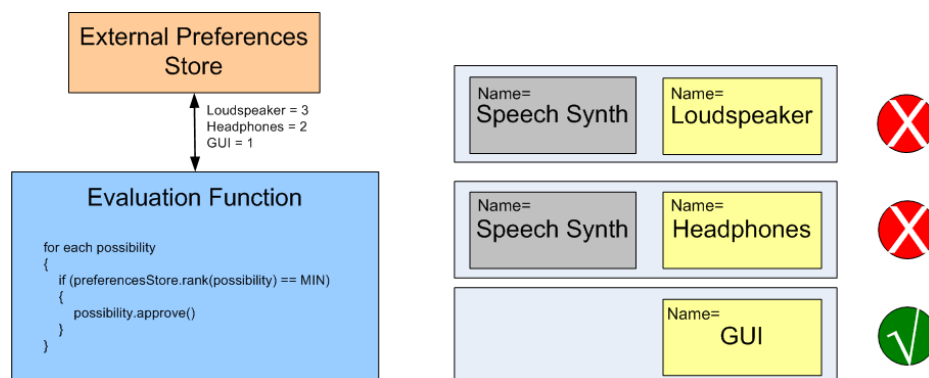
Figure 6.4: *External Preferences Store service being used for ranking*

Figure 6.4 shows an example of this pattern where an evaluation function looks up each possibility within a central "Preferences" lookup service which stores a (user supplied) ranking of components based upon their preferences. A discussion later in Section 6.5.4 covers how multiple users preferences can be combined naturally. This ability to access external services is assumed for many of the later more specific examples discussed in this chapter.

Characteristic 8. *External sources of data are available and can be communicated with via one or more defined interfaces.*

An additional characteristic (Characteristic 8) as a result of this is to provide any external sources of data or services to obtain data as necessary and a mechanism to access them; i.e. a preferences store would be required in the previous example. This does not need to be a single oracle as different evaluation functions dealing with different types of data can be provided with different services providing data and can obtain them via different mechanisms.

For example one implementation of the preferences function described here might obtain the required data via a *preferences service*, as illustrated, while an alternative implementation might instead use a HTTP request to a web service and yet another third implementation might instead prompt the user directly to rank the available options. Note that these different sources, while functionally identical as external sources of data, have very different non-functional properties. Accessing a local database of stored preferences would be quick but may not provide all the information needed or may be dated, remote external databases may be unreliable depending on network conditions and resorting to asking the user may involve a very long delay before the information becomes available (if at all).

There are many different sources of external data and each source of data may have a very different nature. Where the data comes from, how it can be validated and its relationship to the possibility varies with each type of source.

6.4.2.1 Static data

One type of data can be regarded as being static and unchanging within the scope of the application. An example of this data might be determining the physical characteristics of a particular unique device or class of devices (size, weight etc); this information is not likely to change for this device during the lifetime of the system.

An evaluation function, which relies upon static data from an external source, is safe to cache the values internally as it does not require that the data be refreshed at any regular intervals to ensure reliability.

Validation can be performed on static data in a number of ways. In addition to sanity checking that values are within valid ranges (mobile telephones with a weight value of integer '100' probably refers to grams rather than kilograms). In addition, as these values will not change they can be inspected (manually or automatically against a known good source) to verify their correctness and known correct values can be propagated in favour of potentially incorrect values.

Static data may relate to the possibility as a whole or to some subsection of the possibility. For example, a possibility may include references to particular components or physical devices which may have static data associated with them.

6.4.2.2 Sensor data

One particular application of an external source of data in a ubiquitous system is context to enable a context aware or sensitive system. A contextually sensitive system may make decisions on appropriate interaction possibilities to use based on features of the user (such as their location) or on environmental factors which may affect usability of particular devices (such as light levels, heat and ambient noise levels). Other contextual factors that may be taken into account include the physical infrastructure available (state of the components) or the social environment (number and location of guests or visitors within the house for example). In other words; this allows contextual changes based on a variety of different targets. This allows a system to use particular interaction techniques only in certain circumstances.

External data for this purpose can be collected from a sensor, or set of sensors, and interpreted by the evaluation function directly. Examples of external sensor data sources are:

- Optical/Vision - Supply information on light levels, movement detection and presence detection (photo-diode, camera sensor, IR/UV).

- Audio - Background noise, type of background noise, location of noise source and speech recognition (Microphones or arrays of microphones).

- Motion - Detecting motion of users or objects within the environment (PIR, accelerometers, GPS)

- Location/Usage - Position of user(s) or objects within the environment. Use of doors, windows via contact switches. (GPS, PIR, RFID, Contact Switches).

- Personal Sensors - Biomedical sensors for detecting the users state or condition. (Heart rate, pulse, skin resistance, blood pressure).

- Safety sensors - Detecting hazardous or potentially dangerous situations. (Fire, CO_2, intruders).

These external sensor steams may have a direct association with the possibility (i.e. microphones to check ambient noise levels to check if speech recognition can operate correctly) or may actually have no specific relationship to the possibilities being evaluated

other than an association that has been selected by a user at some previous point in time (i.e. a pre-set condition that when the user is in the bathroom use the speakers). In addition, the context data may be used as part of a larger decision making process. This is discussed in greater detail in Section 6.5.4.

Sensor data contextual information is likely to be capable of rapidly updating which prevents local caching of the values. Evaluation functions must therefore have a way of retrieving updates or allowing updates to be pushed to the evaluation function; this is discussed further in Section 6.6.

6.4.2.3 Context Servers

The data collected from external sensors may be noisy, be of poor quality and it may require preprocessing steps before it is useful or to transform the data into a higher level of abstraction (microphone audio -> bandpass filter for particular frequencies -> speech to text -> command interpretation). This would require that processing code be implemented in each evaluation function that wished to use the contextual information. To alleviate this it can be desirable to have one or more sources of aggregated or processed contextual information. For example in the case of multiple sensors being combined to give a more reliable, or informed, analysis of the environment or users status.
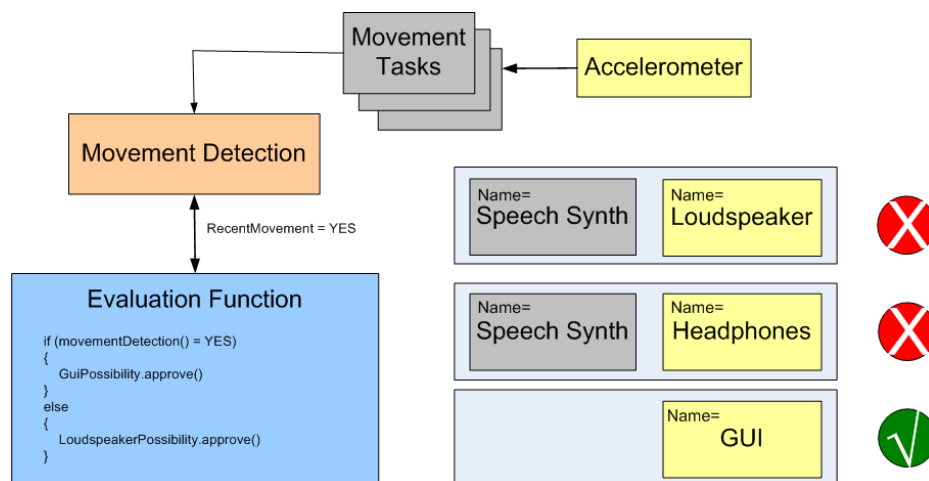
Figure 6.5: *External Movement Detection contextual service being used for ranking*

Figure 6.5 shows a sample contextual system where the evaluation function queries an external source of data, in this case a movement detection context server, to determine the appropriate device to use. There may be multiple sources of context available and these

can be supplied either in a distributed or centralised (aggregated) manner. Context may be available from one or more contextual servers, each of which may be responsible for one or more items of context. Context can be further aggregated by these servers. The modular nature of evaluation functions means that additional sources of context can be supplied in an ad hoc manner as required without the need to design a system in advance specifically to deal with this type of contextual data.

Characteristic 9. *It is possible to provide one or more shared services for preprocessing sensor data which may be further aggregated.*

To implement this within a system requires that additional services can be provided, where necessary, to process or aggregate sensor data which can be accessed and/or updated concurrently by multiple data sources or evaluation functions.

The presence of Context Servers within a contextual system is argued for by Dey and Abowd [193] who classify them into Aggregators and Interpreters. Aggregators act as a gateway between applications and sensors to hide complexity of underlying contextual sensors while Interpreters are responsible for abstracting or interpreting low level information into higher level information.

Data that is supplied by a contextual service may be higher quality, more reliable or at a higher level of conceptual reasoning than data from some other contextual service which may motivate its use. A typical example might be a collection of different movement sensors, the raw data is processed to (i) reliably detect movement, and (ii) combine the results of multiple movement sensors, in order to estimate the users current location. However, it may be that a contextual server involves an additional computational overhead when compared to decision making based on the raw data. For example, a movement detection algorithm may improve its accuracy through sophisticated machine learning or particle filtering approaches, however this may involve significant amounts of CPU or memory, and it may be the case that a possibility that uses a simpler source of context for movement detection may be preferable even where it gives inferior results.

6.4.2.4 Human Interaction

In addition to precompiled static data (Section 6.4.2.1) and sensor data (sections 6.4.2.2 and 6.4.2.3), another source of external data for evaluation functions is to directly retrieve the correct choices and other relevant data from the user.

This approach is only mentioned briefly in this section and will be discussed in more detail
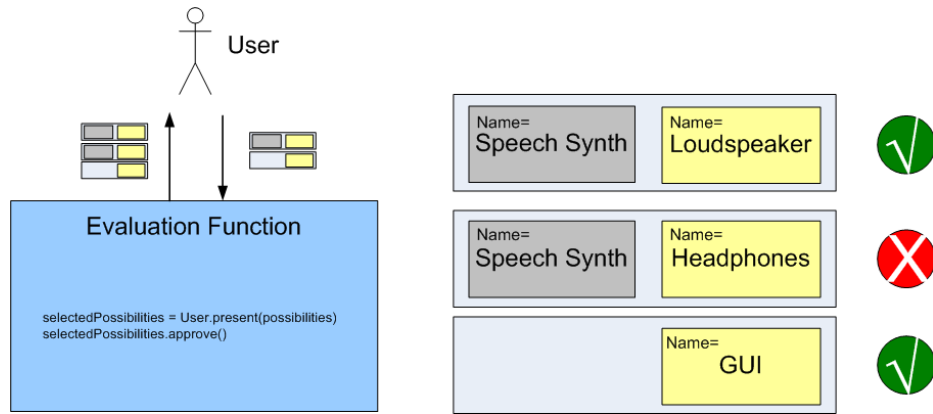
Figure 6.6: *User being directly queried for suitable possibilities*

in Section 6.7.

6.4.2.5 High level / Ontological data sources

In the previous section, only simplistic attempts at combining different sources of sensor data together have been shown. However, a large amount of information may be known about the sensors being used which can be used to infer other properties. For example, to choose to use the audio component which is physically closest to the user; the users location can be determined via suitably processed sensor data as well as having a record of the location of each audio location in the room and it is possible to use Ontologies [55] to infer relationships between components and devices.

The notion of Ontological Service Discovery systems come from the Greek meaning of the term and asks the questions "What things exist?" and "Into what categories can we put them?". An Ontological data model is capable of reasoning about the relationships between the components it contains references to. An Ontological data model of a farm might store the relationship that the entity 'milk' is a type of 'food' which comes from 'cow'. Furthermore, a 'cow' lives in the 'field' which is 'beside' the 'house'. This allows you to perform queries about the existence of foods near the house by joining these individual statements of fact together to perform intelligent queries.

Shown in Figure 6.7, it is possible to try to reason about which sensors / interaction devices might be appropriate based on their purpose or location. The task of monitoring the temperature of the bath water for example might discover a selection of components that measure temperature but only one that has the property of being submerged in water and located at the bath.

94
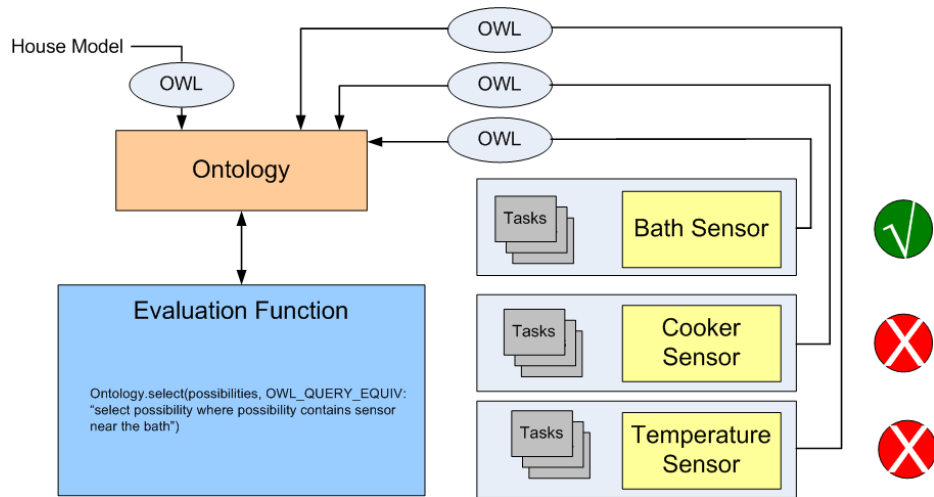
Figure 6.7: *External Ontology Service for reasoning over possibilities*

In order to achieve this within the evaluation function model it is necessary to provide, either integrated into the Service Discovery service or as an additional service, an Ontological model of the components which is capable of accepting relational models concerning the devices and entities within the environment and allows queries from evaluation functions.

Characteristic 10. *For ontological relationship an service must be provided that accepts ontology models from multiple sources and allows relational queries to be performed over the entities available.*

This requires an ontological (or similar) description of device features, location and purpose to be provided to an ontological reasoning engine. This engine must furthermore provide an interface that can be used to perform queries upon this to perform reasoning such as determining components meet the requirements. This is described in Characteristic 10.

In this section a number of different sources of data have been discussed which can be used by evaluation functions to reason about the Targets. In this model, there can be multiple disparate sources of information in use at one time. These sources may overlap in functionality but may differ in non-functional regards such as response times, reliability or computational requirements. It is possible for evaluation functions to use the most suitable source of information that is most suited to their needs.

95

6.5   Means

This section discusses the ways in which an evaluation function can process information, about targets, from the sources discussed in the previous section and then goes on to discuss some of the properties of evaluation functions (Persistence and Combination) which allow for more powerful modelling of requirements.

The paper by Thevenin and Coutaz describes features such as the system task model and system rendering capabilities which are actually used to perform adaptation, as being part of the Means axis. However, this concentrates on features which are likely to influence the choice of appropriate interaction techniques. The basic characteristics of this approach were discussed in Section 6.1 of this chapter while Chapter 7 contains a discussion of tasks and their relationship to evaluation functions and specifically Section 7.2.3 provides a detailed description of their implementation.

6.5.1   Analytical / Custom

The first category of processing introduced is a type of evaluation function which is designed for a particular analytical role. These may be custom designed and built to perform one specific function or role.

It is possible to devise algorithms or formulas that can be applied in particular circumstances to select output devices by from a set of possibilities by application of the algorithm to a fixed set of input data (obtained from one or more sources as discussed in the previous chapter). Such reasoning might compare a possibility to previous quantitative or qualitative research that provides aggregate data on the "best" possibilities to use. An example of this is research by MacKay et al. [130] who performed research on the best presentation styles to use to present textual and visual information (web pages), the presentation layout engines are shown in Figure 6.8. MacKay found some interesting results such as the fact that users liked a direct view style whilst in reality they performed poorly with it.

The results of this study can be applied in a variety of different ways - so there may be many different evaluation functions which use this data - and implemented using different algorithms to achieve different goals; for example one evaluation function could attempt to minimise error rates, while another could maximise bandwidth (information delivered to the user per second) or to maximise the users satisfaction.

Another example is the SUPPLE [231] system which is used to layout a dynamic GUI. SUPPLE assigns costs to each available widget that could be used in a GUI (where each

Figure 6.8: *Gateway, Linear and Direct layout engines from MacKay [130]*

widget could be represented as a possibility within this approach) and has an algorithm for deciding which one to use based on the costs. The algorithm that evaluates the costs can be implemented as an evaluation function. The algorithm as presented in SUPPLE uses fixed costs and weightings for particular possibilities but the costing data could be obtained from external data sources.

The COMET [32] system is similar to SUPPLE in that each component has a preassigned suitability value to particular contextual situations which are used by a fixed algorithm to decide which of the components should be used. Figure 6.9 shows this approach implemented using evaluation functions.

Again, this evaluation functions can operate using both data from the possibilities or by accessing external data sources to determine the value to use for a particular possibility. Analytical functions require a source for obtaining these metrics that may be analytically computed from some algorithm (Sousa) and stored internally, or stored in an external datastore as in SUPPLE, or access to components directly (or some veneer for the components) such as in COMET.

More specific and targeted evaluation functions can be created to fulfil specific purposes. An example could be the "follow the user around the house" evaluation function which selects the audio component closest to the user. In this case it may be necessary to parametrise the evaluation function with the user to be followed or with the source of movement or location data.

In each of these approaches it is necessary to be able to specify evaluation function behaviour (algorithms) - i.e. in order to be able to program evaluation functions. This requires that a developer can provide evaluation functions written for a specific purpose
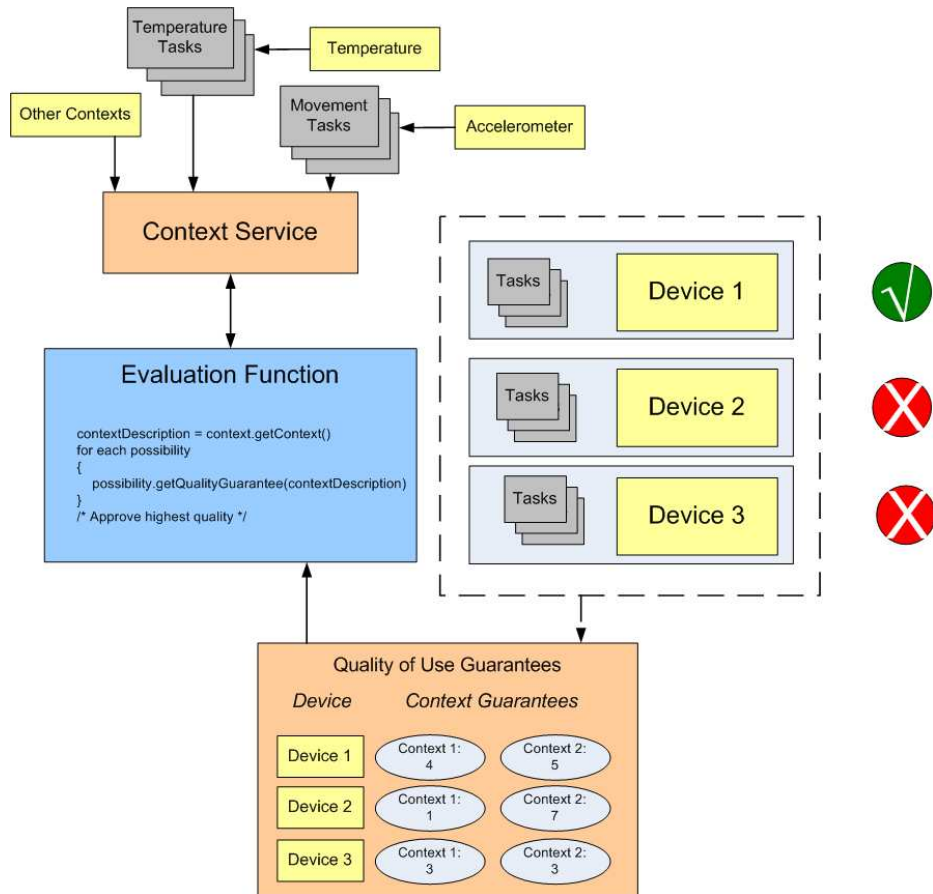
Figure 6.9: *Each possibility is an introspective agent (as in COMET) which can calculate introspective quality of use guarantees given a provided context*

which can be selected and used by a user. Additionally, evaluation functions may need to be parameterised depending on the requirements of the algorithm.

In order to make these usable by end users it is necessary to be able to load evaluation functions represented as software objects or modules into the system dynamically to make these available without needing to redeploy the system. For example they may be downloadable from an online catalog. Experienced users or developers may be able to design their own analytical evaluation functions and submit them to a community supported library of functions.

To enable this, analytical evaluation functions can be programmed in a suitable Object Oriented programming language and dynamically loaded into the running system. This once again relies on Characteristic 3 to allow dynamic loading of components into a running system.

6.5.2 Policies

An alternative approach is to allow for the use of more generalised Policies or rule based reasoning [113] to decide which interaction components to use. A typical policy might take the form of "IF time_after(7am) USE gui ELSE USE speaker". This is useful in circumstances where the required behaviour is both well understood and complex. This may result from situations where the doctor or health professional specifies a care regime which in turn specifies particular choices of interactive components to be used at particular times of the day.

Policy or preferences requires considerable forward planning on which devices are to be used and are typically created by expert users - such as the doctors or health professionals in a home care scenario. The need to specify the result of every possible eventuality as a rule can result in large and difficult to maintain rule sets which may result in conflicting policies. Ongoing research on the challenges of detecting and resolving conflicting policies [229] aim to address this limitation.

The complexity of policy systems does raise a number of questions about who is in charge of configuration. Often a monolithic policy system removes the ability of the people actually using the system to make changes without consultation with technical or administrative help in order to implement the changes. Policy changes may be driven by assessment of conditions by doctors or home care authorities and end users may not be aware of how to initiate changes themselves.

The integration of policies within the evaluation function model helps to control these problems by (i) providing alternatives to policy rulesets for users who would prefer not to use a policy language, (ii) allowing action abstraction of policy targets and conditions, and (iii) maintaining the ability to use policies for static well understood domains or advanced users.

Characteristic 11. *A policy engine is available with allows for the specification, manipulation and execution of policies in a policy specific language.*

There are two approaches that can be used to integrate policies into the approach described in this work which can be used either individually or jointly.

The first approach is to specify within the policy language that evaluation functions can be a valid target or action of a policy such as an action or task that needs to be performed. That is to say instead of specifying a specific component or collection of components as part of the policy rule it is possible to instead use a textual representation of, or

reference to, an evaluation function which is executed instead of using a fixed specification in the policy. An example of this would be the policy `IF time_after(7am) USE Result_of_Evaluation_Function_A() ELSE USE Result_of_Evaluation_Function_B()` and this is referenced in Characteristic 12.

Characteristic 12. *Evaluation functions can be the target of a policy.*

This can be imagined as a form of action abstraction. Instead of supplying a concrete choice the system should choose the best alternative in the given circumstances. This allows the use of the other techniques discussed in this chapter without having to complicate the underlying policy language with notions of context, external data sources, state etc. and allows addition of new approaches without a re-specification of the language. The level of complication of the policy language and the boundary between Evaluation that occurs within the policy engine and the evaluation function approach is interesting but not discussed here.

The second approach allows evaluation functions to be used to enforce policies within the system. For example, the policy manager may only be a collection of policies to specify the aims of the user(s) and an evaluation function is responsible for then reading the rules and attempting to enforce the policy rule set through its selection of possibilities to use. In effect, the policy (or group of policies) becomes an evaluation function.

Characteristic 13. *Evaluation functions can execute policies during evaluation of possibilities.*

Characteristic 13 requires that evaluation functions are capable of accessing the policy management system and either executing policies on its own or executing them via the policy manager. Figure 6.10 shows an example with both of the previous approaches in use where the policy engine not only uses evaluation functions as a target for a policy but where the evaluation function is capable of querying the policy manager in order to determine which of the possibilities it should select.

This integration requires a suitable policy engine plus a mechanism for specifying policies, either as an application interface or as a language specification, which the policy engine can interpret and which can be accessed by evaluation functions.
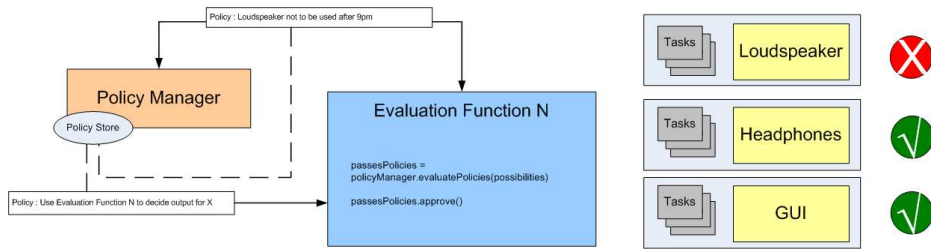
Figure 6.10: *Policy Manager shown with both use cases: (i) arbitrating the choice of evaluation function, and (ii) As an external data source to be used with evaluation functions*

6.5.3 Persistent functions

The previous examples have treated evaluations as polled services, which are queried when needed, but it should be made clear that there are strong advantages in designing persistent objects which have a longer lifecycle and can be queried again later.

A persistent object (in this case an evaluation function) can monitor the state of the evaluation it was previously asked to perform and if the state changes it can notify the appropriate components in order to signal a re-evaluation. This approach is particularly useful within the domain of context sensitive functions as it means that an evaluation function can be given the job of ensuring that the currently selected output device is updated continually according to the current context. This makes possible situations as shown in Figure 6.11 where an evaluation function is tasked with playing music to the user by following them around their home and using whichever audio devices are nearby.
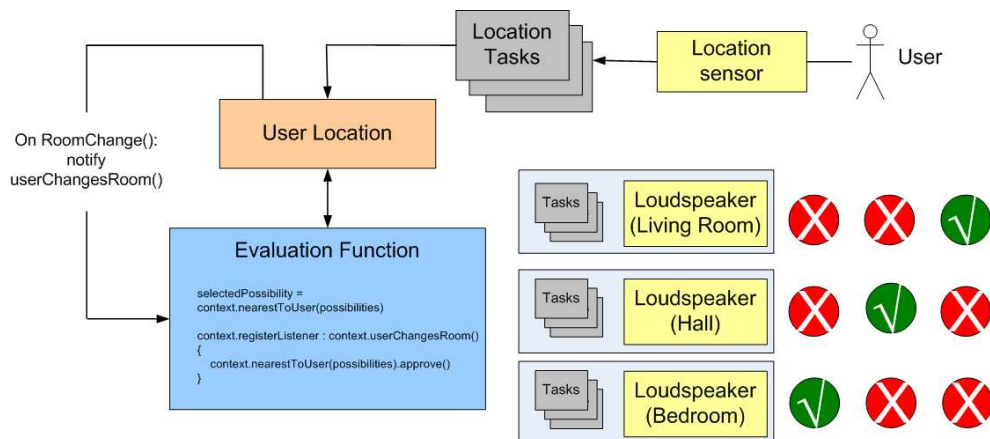
Figure 6.11: *Re-evaluation of Context based evaluation function over time as Context changes; the user moves from the bedroom to the hall to the living room*

By representing evaluation functions as persistent objects that can be manipulated within an adaptive system this allows a particular instance of a persistent evaluation function to service multiple interaction tasks by allowing the same object to be reused.

To do this effectively the evaluation function needs to be parameterised to identify the source of the request in order to allow it to vary its behaviour based on the caller. A typical example of this functionality in use might be to implement locking functions over restricted resources - "losing" evaluations could be directed to use a lesser quality configuration as shown in Figure 6.12.
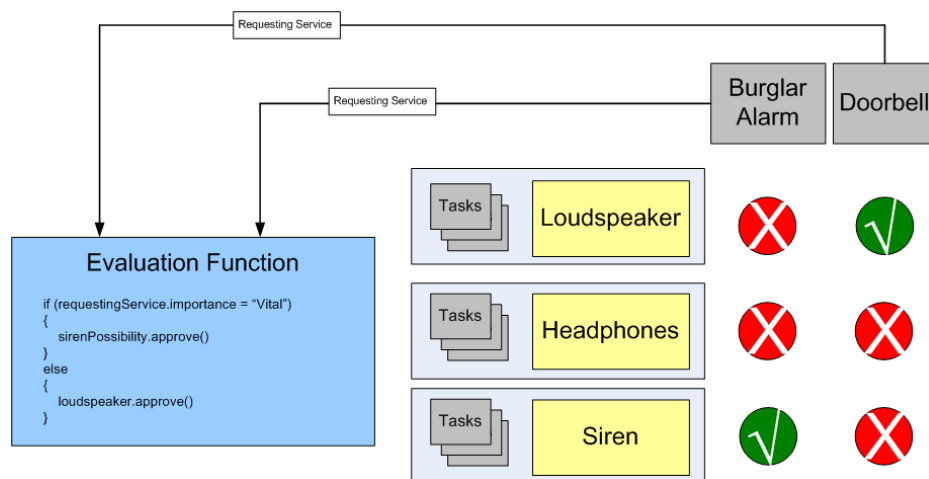
Figure 6.12: *A single evaluation function being used to choose different output devices based on the requesting service*

An extension of this which is made possible by the stateful nature of the function is to allow for an important device (e.g. loudspeaker) to be taken over by an important task (such as announcements or emergency notifications) but allowing its use otherwise.

Stateful functions can be used whenever you wish to enforce a particular behaviour over multiple tasks or where two tasks must cooperate on the choice of interaction technique. This statefulness, the ability for multiple tasks to use a single evaluation function and the ability of the evaluation function to be parameterised with a reference to the calling task allows an evaluation function to make decisions regarding the possibilities to select based on the needs of all the tasks it is used by.

Characteristic 14. *Evaluation functions can be made into persistent objects, can contain state and be called by potentially many parent processes that can parametrise the evaluation function to identify which process requested an evaluation.*

In order to implement persistent and stateful evaluation functions, it is necessary to provide evaluation function instances with a location to store data that it can use to track its current state - this is most easily achieved by implementing evaluation functions as first class objects within an object oriented programming language. Characteristic 14 summarises the required features for persistent evaluation functions. Additional features, particularly relating to temporal behaviour, made possible by stateful evaluation functions are described later in this chapter - see Section 6.6.

6.5.4 Combining Evaluation Functions

In this chapter, evaluation functions have been described as singular entities encoding a single purpose or measure of a target. It is often desirable, or required, to incorporate the results of multiple criteria within a single evaluation. Instead of being forced to create new evaluation functions for each combination of criteria it is desirable to be able to take individual criteria, modelled as evaluation functions, and combine them together. This is included in the model presented in Chapter 5 which includes the ability to combine evaluation functions together in order to form more powerful evaluation processes out of smaller components.

Ideally, it would be possible to devise a single and uniform semantic method to combine results from evaluation functions. However, there can be no general solution to this problem. Consider any adaptive system which attempts to combine the preferences of a number of users ($N > 1$). Each of the users has a preference for which of the many different possible devices or combination of devices should be used ($N > 2$) and these preferences should be combined to create a selection of which device or devices should be used.

More formally, the aim is to extract a preference order for a given set of outcomes where each decision criteria has a a particular order for the outcomes and to derive a method which transforms this set of preferences into a single global preference order.

In social welfare theory, Arrows impossibility theorem [7] demonstrates that systems meeting the criteria above and which have three or more outcomes (candidates/possibilities) are subject to a series of limitations in respect to a certain set of criteria. Arrows theorem considers the following criteria which are described as reasonable requirements of a universal welfare function:

Dictatorship: The system should take into account multiple voters wishes and should not simply return the results of a single voter.

Universality: For any set of individual preferences, there should be a unique, complete (considers all input preferences) and deterministic global ranking.

Independence of Irrelevant Alternatives: Ranking a subset of preferences should have the same ordering as they would appear in the entire set. Briefly, adding irrelevant alternatives should not affect the ordering of the other candidates (vote spoilers).

Pareto Efficiency: If every criteria prefers a certain option to another then it should be ranked higher in the resulting ranking.

Arrows impossibility theorem shows that any system which satisfies the Universality, Independence of Irrelevant Alternatives and Pareto Efficiency must also be a Dictatorship and that it is therefore not possible to maintain these criteria simultaneously within any system which satisfies the description offered previously.

Futhermore, a related theorem by Gibbard [90] and Satterthwaite [195] applies a more general restriction to the problem of selecting a single outcome winner from a set of preferences. The Gibbard-Satterthwaite theorem provides that for any deterministic approach for selecting a single winner from three or more that the one of these three criteria criteria must hold:

Dictatorship: The approach is dictatorial.

Non-Completeness: There is some outcome that can never win. Or,

Tactical-Voting: The approach is susceptible to tactical voting. Specifically, there exist situations where voting down an outcome can increase its chance of winning and there are therefore incentives for a criteria to lie about its preferences.

As a result of these two theorems, *it can be concluded that there is no general solution to the problem of selecting one or more winning outcomes from more than two available outcomes given a set of preferences*. This is a general problem for all adaptive systems and is not restricted to the evaluation function approach described in this thesis.

Pragmatically, despite the lack of a general solution to this problem there are a number of different approaches that can be adopted to allow combination of preferences. These approaches to combining information from more than one criteria (or evaluation function) operate by weakening or removing one of the criteria listed above; a compromise which is necessary for a functional political system. For political voting systems these are most commonly violated criteria are Tactical Voting, Independence of Irrelevant Alternatives and Pareto Efficiency.

The remainder of this section will focus on example combination functions with different

semantics as well as discussing which compromises they make in deciding upon a winning outcome. These examples should not be inferred to consist of a complete set of all possible approaches. Following this, a unifying model for combining these approaches is given which allows a standard form for specifying and using these combination functions within the evaluation function model.

6.5.4.1 Voting

Imagine that an evaluation function as described in previous sections is analogous to a voter in a political election. Each evaluation function or voter has a preference on which possibilities are used or who is elected to office and the final result of which possibilities or politicians are used is an amalgamation of these votes in some predetermined fashion.

In the trivial case there is only one evaluation function making the decisions or allowed a vote - in this case the system is equivalent to a dictatorship. Otherwise, a voting algorithm is required to combine the results. Many of these have been extensively researched in the political literature. Some of the most common systems for single winner voting are summarised below.

Plurality: The plurality method is simplest voting method and is often used to elect heads of state. In this system, each voter makes a single vote and the winner is the candidate, or possibility, with the highest number of votes.

Approval/Disapproval: Approval voting is where a voter can vote for as many candidates as they like but that they can only indicate approval or disapproval of a candidate.

Ranked: Ranked (range) voting allows voters to specify a rank to each candidate, or possibility, in order of preference and the candidate with the highest score wins.

Condorcet: A condorcet method employs a pairwise comparison of all candidates based on ranked list of candidates provided by the voters in an imaginary one on one comparison.

Instant Run Off: Candidates are voted for in order. If no candidate wins by a majority then the least popular candidate is eliminated and voters who voted for that candidate have their second vote counted.

In the above list all the approaches violate the Tactical Voting criteria and additionally ranked (range) voting violates the Pareto Efficiency criteria while all others violate the Independence of Irrelevant Alternatives criteria.

Similar systems exist for multiple-winner systems such as Proportional Representation or Semi-proportional Representation based systems. Single Transferable Vote is a multiple-winner system which is analogous to the Instant Run Off single-winner system while Plurality-at-large is the multi-winner equivalent of Plurality.

Approaches based on real political voting systems have the advantage of familiarity for most adult users due to previous experience with voting systems; although some voting systems can be extremely complicated for a participant to predict the result of their vote.

6.5.4.2 Set combinations

Another approach which can be applied to approval votes only (yes/no preferences) is based on the notions of sets. In set theory binary operators are defined which operate on two sets of outcomes and produce a third set which is the result of a comparison between the two sets. Set notation can be therefore be used to define a series of operations to occur upon a collection of sets in order to create new sets representing the results. The approach of using set notional upon a list of approved vs. disapproved votes would typically violates the universality criterion. A list of common binary operators on sets is described below.

Union: The union of two sets is the set comprising all results that are in either, or both, queried sets. This would allow users to specify operations such as "[Bobs Devices] UNION [Jills Devices]" to specify that all devices specified by either party should be used.

Intersection: The intersection of two sets is the set comprising all results are in both queried sets. This allows a more specific operation to require that a device was in both Bobs and Jills list of devices, i.e. "[Bobs Devices] INTERSECTION [Jills Devices]".

Complement: The complement (set difference) is the set of all members of one set which are not in the other set. This allows for possibilities to be removed from a set for example "[Devices in the kitchen] COMPLEMENT [Devices with audio]".

Symmetric Difference: The symmetric difference (XOR) is possibilities which are a member of one or the other queried set but not both. This allows you to specify queries which would prevent a message being received via a shared device i.e. "[Bobs Devices] XOR [Jills Devices]" would result in the set of devices which only one person listed.

Combinations of multiple set operations can be created. For example "[Bobs Devices] INTERSECTION ([Bobs Devices] XOR [Jills Devices])" would result in a list of only

devices which Bob has approved but which Jill has not - in this case this is actually the same logical result as "[Bobs Devices] COMPLEMENT [Jills Devices]" which demonstrates the ability of set notation to build higher order operations from simpler building blocks.

One popular language in frequent use within computing already is the Structured Query Language (SQL) which uses ideas and methods borrowed from set theory in order to manipulate and query databases. A similar language could be created here to allow for specification of set operations to be conducted which could then be transformed into the appropriate tree structure.

However, due to the size, complexity and inconsistency of the SQL language between implementations it might be advisable to use a graphical user front end for composing set relations by building a tree of operations. This would help users create compositions as it would be possible to display venn diagrams, or similar notations, to graphically display an abstracted version of the result.

6.5.4.3   Functional Perspective

Another perspective should be familiar to developers of software applications and models combinations as part of a functional language which allows arbitrary operations to occur. This approach relies on the observation that evaluation functions transform the traditional role of configuration from one of static values to a function that is used programmatically - similar in nature to the task of writing computer programs.

An approach to implement this is to implement a recursive descent parser which can interpret a combination of evaluation functions described textually and construct an appropriate evaluation function tree from this.

Evaluation Function Tree Example. *Example tree structure of a selection of evaluation functions*

```
f ( params )
{
        g ( params )
        {
                h ( params )  { }
                i ( params )  { }
                j ( params )  { }
        }
        k ( params ){ }
}
```
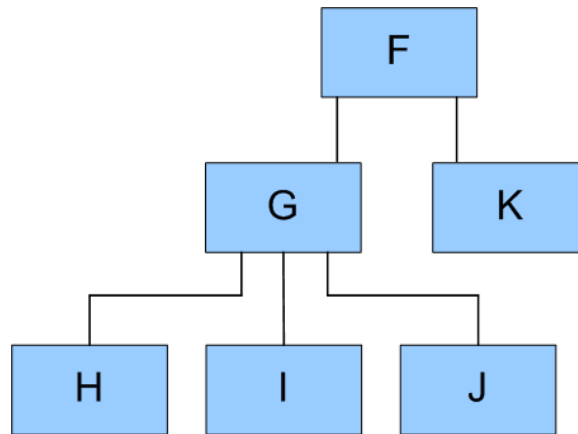
Figure 6.13: *A graphical representation of the tree structure in the evaluation function tree example*

Given evaluation functions `f()`, `g()`, `h()`, `i()`, `j()` and `k()` it is possible to construct a textual description, shown in the evaluation function tree example, representing the combination of these functions represented in Figure 6.13. In this example `f()` is the root function which contains `g()` and `k()` as immediate children with `g()` containing `h()`, `i()` and `j()` as further children.

Characteristic 15. *A language specification exists which allows the description of evaluation functions and their tree structure. Instances of configurations in this format can be interpreted (by recursive descent parser or otherwise) and formed into an evaluation function tree.*

This process relies upon the existence of a suitable parser which can read in descriptions of evaluation function trees in this format and create the corresponding tree.

At this point the concept of parameters for evaluation functions is introduced; evaluation functions have been described in the previous sections which are capable of performing reasoning based on context, policies or sources of external data. Rather than creating one evaluation function for every conceivable action it is advantageous to can create generic evaluation functions for many tasks which can be parameterised to a specific function.

One example of this is the contextual evaluation function described in Section 6.4.2.3. Where a system might have many different types of context, it is often possible for the same logic to be reused but to merely parametrise the evaluation function to change which piece of contextual information it considers when performing the evaluation. This allows a single evaluation function to be reused many times (and even multiple times within a single

evaluation function tree).

Characteristic 16. *Evaluation functions can have parameters that can be used to coerce general functions to perform a specific role.*

If the service discovery subsystem is capable of listing the known required and optional parameters (including types) that an evaluation function is capable of interpreting then this allows for runtime checking that evaluation functions have been supplied with all the required parameters.

Characteristic 17. *Parameters to evaluation functions can be checked by comparing them against the parameters known to be accepted by that evaluation functions description within the Service Discovery system or in its method signature.*

Functional approaches to combination of criteria could prove to be more powerful than either set notation or voting systems as they have the capability of operating over disparate types of input preference data. One example of this would be a function which combines the results of an approval preference and a ranked preference. This might be the case where the approval preference supplies a list of devices which are allowed to be used at any one time (perhaps supplied by a doctor) while the ranked list represents the preferences of a user.

Although evaluation functions have consistently been referred to as being functions it should be noted that they are not restricted to being pure functions in the sense of mathematics and may have side effects - that is to say, the function can have modifiable state or has observable interaction with the outside world. This should be clear from the discussion in Section 6.4.2.

In this section a small language for specifying the combination of evaluation functions has been described which can be easily implemented using a recursive descent parser and an example of how it can be used to combine two preferences based evaluation functions together to emulate an IF statement has been provided. This language can be implemented using recursive descent parsers or as a heavier weight solution respecified as an XML schema which is equally suited to describing tree structures.

Even if this language is never presented to users it can still be used internally to represent the evaluation functions. Naturally because of the complexity in learning to programming languages [53] any functional language used to describe combination of evaluation functions which is presented to users directly would require advanced users.

6.6 Time

This section will discuss some of the temporal aspects of configuration. A number of evaluation functions have already been discussed which have, until now, treated as being responsive only - in the respect that they are queried to obtain a result which they return immediately - but where their results are likely to change over time. Examples of these are the preferences and context sensitive evaluation functions.

During this section, a number of different patterns of temporal evaluation will be introduced and a consistent method of integrating each of these described so they can be used cooperatively.

6.6.1 Queried evaluation

The first temporal pattern discussed will be referred to as Queried evaluation. This is the temporal pattern which has been assumed in previous examples where evaluation functions are called from a process using an evaluate method on the evaluation function. This is a "pull" approach to requesting evaluations to take place. The evaluation function is specifically requested to provide a response, the calling process "pulling" the results from the evaluation function.

At this point the concept of an *Interaction Manager* is introduced. The Interaction Manager is a process which is responsible for coordinating evaluation function calls - the Interaction Manager can take the place of the calling process for an evaluation function and can implement the remaining features of the model such that each calling process does not need to implement the entire model. When a process wishes to configure an application task to use a particular evaluation function, the configuring process should not be burdened with the need to wait until the task has reached the stage where it needs to interact with the user until it can execute the evaluation functions. Nor should the application task itself be concerned with the evaluation process that is going on in order to decide how it should communicate.

Characteristic 18. *An Interaction Manager service which is responsible for querying evaluation functions and setting up the chosen possibilities on behalf of applications tasks.*

The application task should be able to request an interface which allows communication with the possibilities that have been determined to be used with that task; as a result of an evaluation over the evaluation function (or multiple evaluation functions) assigned to

110

that task. The intermediate process of executing the evaluation function and setting up the possibilities is delegated to the Interaction Manager service.

It should be noted that an evaluation function does not need to be executed immediately to return a configuration; rather, an evaluation function should only be executed when the results of the evaluation are actually required. This is an important point as the situation, and therefore results of the evaluation function, may change between the time the evaluation function is set and the time the interaction actually takes place.

Instead, the first time the task requires to communicate it should signal the Interaction Manager that it is now time to execute the evaluation function. When this method is called, the Interaction Manager will execute the evaluation function and receive a list of possibilities.

Characteristic 19. *The Interaction Manager returns a proxy object for communicating with the possibilities that are selected.*

In order to allow for multiple possibilities and re-evaluations, which is discussed in the next sections, the Interaction Manager should return a proxy object for communication with the possibilities to the task rather than returning the actual possibilities themselves. This allows re-evaluation to take place without having to negotiate with the task to conduct a replacement of the possibilities in use.

6.6.2 Timed re-evaluation

An extension to the previous approach which allows re-evaluation of evaluation functions to occur at a later point in time is to define a period of validity of each evaluation and when this expires is to reexecute the evaluation function and update the proxy object that was returned as a result of Characteristic 19.

A centralised approach to implement this is to create a timer within the Interaction Manager which is initialised when the evaluation function is first executed and set to a predefined polling interval which signals when the evaluation function should be reevaluated.

A more decentralised approach is to allow third party services to signal that re-evaluation of an evaluation function is required. This would allow re-evaluation periods to vary for different evaluation functions such that rapidly changing functions could be reevaluated more frequently than those which change rarely. This requires the implementation of a re-evaluation timer located external to the Interaction Manager which then calls a method on

the Interaction Manager to signal that an evaluation function should be reevaluated.

Characteristic 20. *It is possible to signal that an evaluation function should be reevaluated after a period of time has elapsed. The signal can originate from timers stored within the Interaction Manager, within evaluation functions or within third party components.*

The disadvantage of this timing-only based approach is that it does not allow you to reevaluate an evaluation function in response to an event or stimulus, nor does it allow a timed evaluation to be deferred to a later point in time. An alternative approach follows that does allow these features as well as supporting the timed re-evaluation technique if it is desired.

6.6.3   Stimulus-based re-evaluation

In order to address this problem the definition in Characteristic 20 can be extended to allow for signals for re-evaluation to originate from sources other than timers to create a "push" style approach to re-evaluation. This means that the Interaction Manager can be notified by an evaluation function or a third party that circumstances have changed sufficiently to warrant a re-evaluation.

The benefit of stimulus-based re-evaluation is that it allows the choice of possibilities used to be varied in time in response to some external events. A typical example of this might be a context sensitive evaluation function which selects the closest audio device to your current physical location to play your music to you - ideally you would want such a system to be able to "follow" you around the house, automatically selecting new devices as you get closer to them.

Characteristic 21. *Evaluation functions and third party components can trigger re-evaluation of evaluation functions in response to stimulus.*

To do this, assume that the evaluation function is capable of monitoring the state of the external event it is concerned with - this should be a fair assumption as evaluation functions have been described as persistent objects within the system and shown how they can retrieve elements of external data as necessary - as such it should be simple for an evaluation function to register to receive updates or periodically poll an element of context to receive updates.

Characteristic 22. *Evaluation functions and third party components can monitor state of*

external events in order to detect stimulus for re-evaluation.

Once an evaluation function has detected or been notified about a stimulus it can notify the Interaction Manager to reevaluate the evaluation function trees that it is a member of; as of course the results of this evaluation function may be combined with other evaluation functions within the tree - necessitating the entire re-evaluation.

Characteristic 23. *The Interaction Manager can attach a listener/signaller to an evaluation function to receive notifications for re-evaluation.*

Timed re-evaluation can be implemented using this approach by allowing evaluation functions to use timers as a stimulus to initiating re-evaluations.

6.6.4   Deferred re-evaluation

One penalty to re-evaluation that can be observed is user difficulties if the criteria for evaluation change frequently - so frequently that the changes between devices happens more often than the user can keep up with. Additionally, a re-evaluation may occur at a time that is inconvenient for the user where they would actually prefer that the change happens at a later time.

To account for these situation it is necessary to be able to *defer* re-evaluation until a later point in time and to restrict the number of re-evaluations that occur. In addition to allowing evaluation functions to moderate their own behaviour - this behaviour can be controlled using other evaluation functions.

When an evaluation function is assigned to a task the Interaction Manager registers a listener with a reference to itself in order to be notified whenever a re-evaluation is signalled. Instead of, as in the previous section, evaluation functions directly notifying the Interaction Manager it is specified that evaluation functions should notify their parent within the tree which can then opt to either pass on the resulting request to its own parent or to discard the request.

This allows for requests for re-evaluation to propagate through the tree of evaluation functions until they reach the root and eventually the Interaction Manager which then issues the reevaluates the root evaluation function.

Characteristic 24. *The listeners that the Interaction Manager uses are attached only to the parent evaluation function. evaluation functions can attach listeners to their child functions which allow them to decide what to do should a child request re-evaluation.*

As branch node evaluation function within the tree can override and thereby control the re-evaluation requests of the evaluation functions of its children this allows the branch evaluation function to ignore requests from its children if they do not fulfil some characteristic (happened too soon after the last request or did not happen during a defined window of time) or to delay the request until a later point in time (deferring it until the defined window of time) and then reissuing the signal itself.
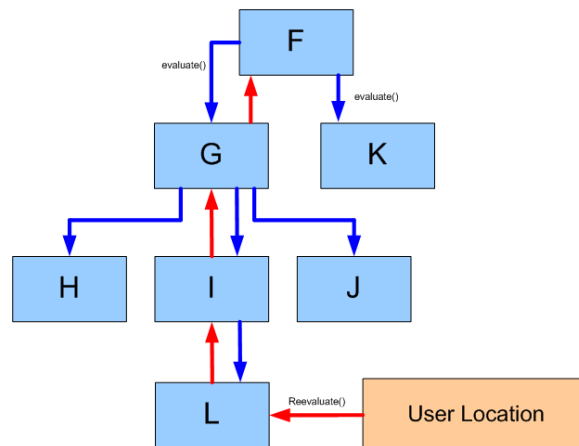
Figure 6.14: *A re-evaluation request is initiated as a result of a change in the users location. This request propagates upwards (red) through the tree until the root node before diffusing among the entire tree (blue) to cause a re-evaluation. Propagation or diffusion can be halted by relevant evaluation functions not passing on the appropriate message.*

Using this approach needless or unwanted re-evaluations can be avoided as the re-evaluation itself does not take place until an entire portion of the tree (from the initiating request node through all branch nodes up to the root node) agrees there should be a re-evaluation.

Evaluation functions can be implemented which are specifically concerned with behaviour of deferment, which can be inserted at any point in the tree to control, propagate or squash re-evaluation requests. This means that leaf node evaluation functions can be focused on their own specific task rather than trying to implement control functionality into them - simplifying their implementation - and have control of deferment in dedicated evaluation functions.

The second addition is to explicitly support the ability of evaluation functions to declare that they are unable to provide meaningful evaluation at this time instead of returning incomplete or invalid results.

Characteristic 25. *Re-evaluation listeners can indicate a deferment state (exception)*

which represents no decision has / or can be be made by this evaluation function. This exception can be caught by parent evaluation functions to implement fall back behaviour.

This can be done by extending evaluation functions to allow them to throw a Deferment Exception. This exception, when thrown, indicates to the calling process (parent evaluation function or Interaction Manager), that an evaluation is not possible at the current time - either due to resources it needs being unavailable, inability to perform some action that is required or by some other reason.

Processes that can call evaluation functions capable of throwing this exception will need to be capable of dealing with it when it happens. For example a parent evaluation function may execute the first of its child evaluation functions and execute the second, and any subsequent functions, only if the first function deferred. Alternatively, the function may instead propagate the exception up the call stack to its parent functions.

Once the deferment situation has passed the evaluation function that deferred may now be able to make a decision on which possibilities should be used which it can indicate by triggering the re-evaluation signaller which would be handled normally.

Evaluation functions that declare ability to throw deferment exceptions should not normally be added as children to other evaluation functions which are not capable of dealing with that exception - however, there is no such inverse restriction on adding functions which do not declare deferment to be children of those functions which can accept evaluation functions which defer.

This approach allows implementation of reactive (response to external stimuli) and proactive (response to internal stimuli) evaluation functions which can be called either once only (singular call to the evaluate method), queried on a regular basis (pulling configuration data) or can be reevaluated immediately as soon as there is a change (pushing configuration data).

6.7   Actor

The final axis from the model of Thevenin and Coutaz is the actor who initiates or is otherwise responsible for an instance of configuration. Throughout the lifecycle of a task that requires configuration there may be a large number of events which occur which are identified as opportunities. The action of identifying opportunities for reconfiguration can be be done by any of a number of actors or stakeholders of the system - for example in a multiuser home each of the residents would be an actor. Each of these actors is capable of

both identifying the need for reconfiguration as well as influencing how that reconfiguration takes place.

The two most general classifications of actors are human and machine. This section first briefly discusses machine actors before discussing approaches which attempt to map onto both a single humans intention followed by those which represent multiple human actors.

One particular strength of this approach is that it allows, as shown in the previous section, multiple actors in each category to initiate and influence configuration.

6.7.1 Machine

Previous sections have discussed a number of machine actors in the form of analytical evaluation functions as well as context based evaluation functions, policy systems and ontological reasoning systems. Each of these is a computer system which is capable of identifying a reason for change and signalling that an evaluation is necessary. As these have discussed these approaches already it is only necessary to reinforce the notion that these techniques are actors within the system in much the same way that a user is. The use of evaluation functions as part of a tree structure allows combination of these actors together.

Characteristic 26. *Computational/machine entities may trigger evaluations or re-evaluations.*

One important caveat to consider when discussing machine actors is that they will often be expressing the will of a human, or a collection of human, actors. To be precise it is necessary to regard all evaluation functions as machine actors but consider the degree to which the evaluation function represents the will of a human actor. An evaluation function which is highly disassociated with the will of all human actors (for example a hypothetical evaluation function which randomly selects possibilities) is "more" of a machine actor while an evaluation function which maps directly onto the intentions and will of a human actor would be said to represent that human actor.

6.7.2 Human

The most direct form of human actor within configuration would be to allow the user to choose specific possibilities by selecting them based on some attribute(s). This is equivalent to allowing the user to directly "wire" every component in the system and could be accomplished by allowing the user to use evaluation functions which select based on

attributes of the possibilities as described in Section 6.4.1. This provides the user the maximum level of oversight - configurations will not change without direct intervention to change the attributes used to select possibilities. This approach is analogous to the Jigsaw [109] approach and could be presented as such - evaluation functions representing connections could be automatically created and manipulated behind a graphical user interface.

This model can be extended by allowing the use of task *Templates* which a user can use to create a task and specify its interactions at the same time. This is the approach used by Speakeasy [67] and presents a description of the task plus a number of "blank" fields where the user enters the device to be used in this instance.

It is possible, of course, to extend each of these methods to allow specification or selection of evaluation functions rather than selection of specific devices - allowing this approach to still be used in combination with other evaluation functions.

It may be desired to allow a human actor a high degree of control over the possibilities chosen but still allow the evaluation function to make a "best choice" in any given situation. An evaluation function which stores a record of a users preferences in respect to some set of possibilities such as the one described in Section 6.4.2 would be capable of satisfying this requirement. Such a function can be said to map quite accurately to a users intentions but only if the preferences are either regularly updated or unlikely to change. It is possible that a single set of preferences does not satisfy a large enough number of circumstances. This can be addressed by using combinatory functions, as described in Section 6.5.4, to combine multiple sets of preferences and choose the correct set of preferences based on the current context.

A more direct approach is to ask the user at the point of configuration or at a predefined point in time which possibilities should be used. If the system is about to initiate a long interaction with the user then a prompt to choose the most appropriate technique could be used where a list of possibilities can be presented.

Characteristic 27. *Human entities may trigger evaluations or re-evaluations and evaluation functions may interact with human actors to decide appropriate courses of action.*

It does not necessarily need to be the user who will be the subject of the interaction who decides what the appropriate technique is - for example, it could be that the husband manually chooses an appropriate interaction technique for his wife to use immediately afterwards. This approach generally relies upon the notion, and ability, to defer evaluations until a later point in time - for example it may be inconvenient at this point in time and some

other approach should be used, temporarily, until such a time later on where the user can select from the prompted list.

Some of the advantages of this approach would be the users complete control over interaction while still retaining the ability to update configuration at the point that an opportunity for reconfiguration exists. However, this does require the user to respecify the configuration every time something needs changed and does not support changing the configuration without the users direct intervention.

The user may not only be involved in a "once off" evaluation, one approach is based on the idea of "evolutionary art" [219] which relies upon repeated contact with human decision making. In this approach the user is presented with a small selection of possibilities which they can choose between. Once the user has made this initial choice then further choices, when presented, would be random variations upon the last selected possibility.

Characteristic 28. *Evaluation functions may interact with human actors over a period of time. Either directly or through configuration of evaluation functions.*

In this way the user would "evolve" their selection of possibilities towards those most suitable. When selecting which possibilities to use, the evaluation function would select possibilities which most closely matched the evolved possibility that had been created by the user. By completing more evolutionary cycles the user would improve the accuracy of the selection. These evolutionary cycles could be performed in advance of configuration or as a response to a poor configuration reflecting the need for further evolution.

A concrete example of this approach to defining a user interface is in the work of Masson, Demeure and Calvary based on the COMET system described previously. They present a system called Magellan [139, 140] which includes a user evaluator component, shown in Figure 6.15.

The Magellan system stores the available presentations of a UI in a semantic network which allows for queries such as *Find a presentation for task T that is close to the P presentation*. This allows Magellan to explore the design space of a task by iteratively refining the search space to find available presentations which are closer to the users ideal. Iterations are driven by the user who selects a new variation upon the interface at each evolutionary cycle. The initial UIs to begin editing are generated either randomly or based on a manual specification and each iteration selects new alternate presentations based on similarity to the previously chosen presentations. Incorporation of this approach into the evaluation function model would allow for the selection of possibilities based on a similarity score to previously selected (or currently running) possibilities at each evolutionary iteration.

Figure 6.15: *Magellan: User Evaluator component allowing the user to select the next iteration of the UI*

In order to implement evaluation functions with any direct user interaction, it is necessary to be able to allow evaluation functions to communicate with the user in order to query them to either obtain lists of preferences, ask them which possibilities to use or to present a user interface for evolutionary selection of possibilities. This interface can be presented along with the same interface used to select evaluation functions, or, alternatively can make this configurable by allowing other evaluation functions to be used to select the interface to be used to query the user about possibilities. This can be configured either globally or per evaluation function.

6.7.3 Collaborative

To involve multiple actors within a configuration they can either combine together individual evaluation functions representing their intentions, as described in Section 6.5.4, but it is possible to implement evaluation functions which are specifically designed to represent a group of peoples preferences. These are broadly classified as recommender functions.

Characteristic 29. *Evaluation functions may operate over aggregated preferences or activity histories in order to make decisions.*

119

Recommender functions use an approach based on reasoning where evidence about previous success rates of configuration. A number of recommender algorithms were discussed in Section 2.3.5 which could be employed here. A recommender will often use a form of empirical logging, either based on a single user or a large sample of users, where the success rate of a particular configuration - in this case possibilities - is derived from number of uses of that possibility as well as the periods of use. Here a variation of collaborative filtering or recommending can be performed to remove or highlight configurations that other people have had particular success with. In addition, possibilities which are found in conjunction with other possibilities can be selected in the event that one of them is already in use. An exemplar approach to this is the statistical analysis of previous configurations to determine how well "survived" the configuration is based on how often it has previously been chosen by the user in combination with how frequently it is found in conjunction with existing possibilities.

In addition to recommendations for possibilities, it is possible to use third party recommendations for selection and composition of evaluation functions. Explicit support for collaborative reasoning may be used where the same configuration, in terms of evaluation functions, that another user uses is wanted or copy segments of their configuration that work well already. For example, it might be the case that it is desired to duplicate another persons configuration, or to copy it and then further modify it for their own needs. This collaboration might be facilitated by sites designed to encourage and foster sharing by providing places to gather and encourage sharing of evaluation functions and evaluation function compositions. An example of this might be like the Mozilla Firefox and Thunderbird add-on extension sites [73]. Such a site would make a convenient place for sharing and distributing the evaluation functions themselves.

All collaborative-based evaluation functions would work best with a larger community of people using the collaborative features in order to allow for accurate predictions of possibility suitability to be made.

6.8 Overview

This chapter has presented a large number of different styles of evaluation functions that could be implemented using the approach in Chapter 5. Each of these approaches has been placed on one axis of the model by Thevenin and Coutaz but while each approach here has been presented only on one axis it is not the case that each approach only belongs to a single axis. In fact each approach here is placed somewhere within a four dimensional

higher order space - an example of this is the Human actor (Actor axis) which may be realised through the use of a preferences based evaluation function (External data on the Target Axis), may be persistent (Means), may be combined with other evaluation functions (Means) and may involve stimulus-based re-evaluation whenever the user updates.

The number of categories that an individual evaluation function can be placed into is very large but the Thevenin and Coutaz model has been used primarily to provide structure to the discussion of which evaluation functions are available. There is no claim that this is a complete list - on the contrary the numbers and types of evaluation functions is limited only by implementing developers in terms of scope.

The next section discusses the development of the MATCH system, which includes a number of these approaches in an implemented system - including recommenders, ontologies and context sensitivity. The implementation of several required components is discussed to realise these functions and particular attention paid to the implementation of the Interaction Manager as described in Section 6.6.1.

7

Implementation - MATCH Framework

This chapter presents the MATCH software framework, a software implementation that serves as a proof of concept of the model presented in Chapter 5. The MATCH software framework demonstrates that the proposed approach is feasible and that the resulting system satisfies engineering related criteria identified in the thesis statement (feasibility, scalability, flexibility).

The MATCH framework was built within the larger research effort of the MATCH project

discussed in Section 1.3 and incorporates elements from each of the four MATCH research themes; (i) home networks, (ii) lifestyle monitoring, (iii) multi-modal interfaces and (iv) evolutionary configuration management, although the focus of this work is evolutionary configuration. The area of home care system development targeted by the MATCH project is of particular interest as existing home infrastructures tend to be made up of closed single-function systems composed primarily of monitoring and alarm systems [56] and these systems typically do not interoperate with each other and are not amenable to configuration after deployment.

As such, the MATCH framework includes elements which are not, strictly speaking, necessary for the implementation of the model. Emphasis throughout this chapter will be on the core subsystems, their relationships with each other, and particularly on those elements which are necessary to support evolution.

This chapter will present the framework and show that it is a particularly suitable design to support evolutionary configuration due to good design decisions and incorporation of the process and model that has been presented in previous chapters. This chapter will address the configuration challenges in the overall architecture of the infrastructure.

Section 7.1 presents an overview of the design of the architecture. Section 7.2 provides more in depth details on individual features of the framework. Sections 7.3 and 7.4 discuss the specific implementation of the Interaction Manager subsystem in depth - focusing on its implementation of the model from Chapter 5 - followed by a validation of the approach from a software engineering viewpoint. Sections 7.4.1 and 7.4.3 present applications built within the framework designed to test its feasibility and Section 7.5 concludes.

7.1 Design

The design of the MATCH framework values separation of concerns and modularity which led to a task based design based upon the OSGi (Open Services Gateway Initiative) [138] framework. OSGi is a modular Java framework which provides the ability to access and use a wide variety of different protocols such as X10 and UPnP without the need to write custom drivers. OSGi uses a model that allows modules or subsystems, known as bundles, to be dynamically added and removed from a running Java Virtual Machine (JVM). As OSGi is layered on top of a JVM it continues to allow access to all the native features of the underlying JVM as well as allowing incorporation of native non-Java code via the JNI (Java Native Interface) framework to an OSGi based application. The choice of OSGi makes it easier to incorporate a multitude of disparate devices, allows addition of subsystems at

runtime and encourages modular separation of bundles - all key features to enable dynamic system reconfiguration at runtime.
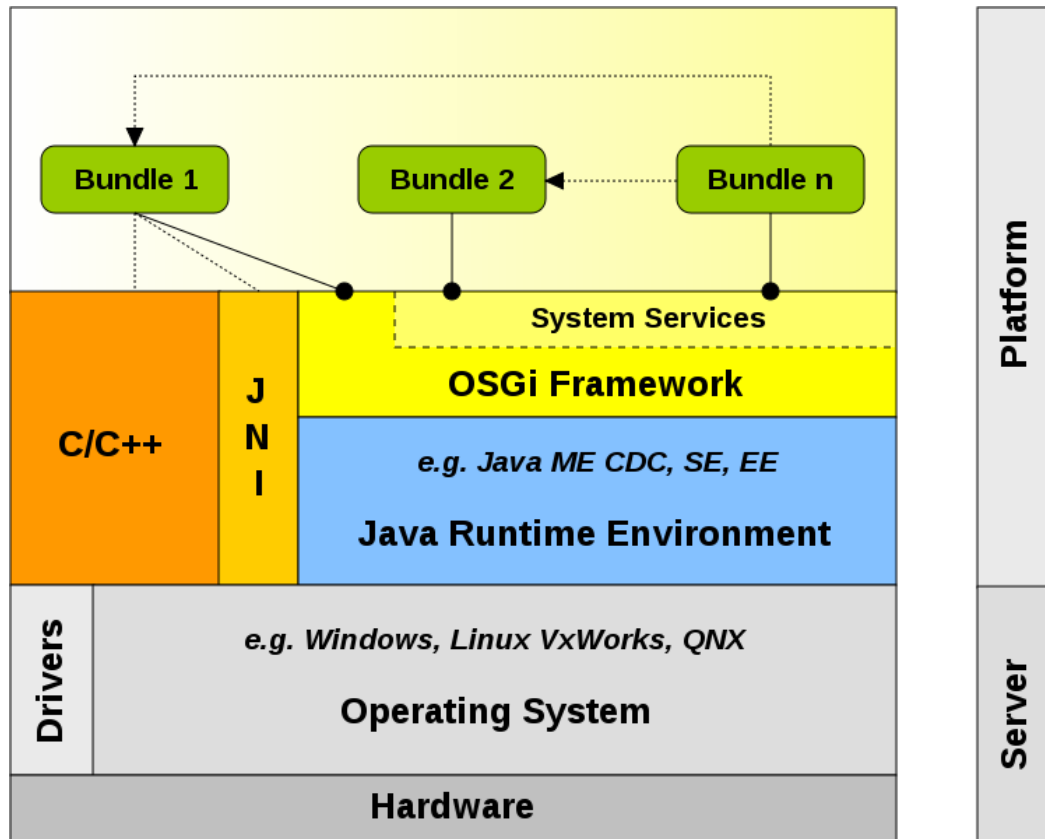
Figure 7.1: *Overview of the OSGi Layering - Image originally by Michael Grammling and Bill Streckfus. Used under Creative Commons Attribution ShareAlike 3.0 license.*

Figure 7.1 illustrates the layering employed in the OSGi framework. At the bottom of the stack is the Operating System and Hardware, which may be a standard PC Desktop operating system such as Windows or Linux as well as embedded devices such as Smartphones or integrated devices in cars.

A Java Runtime Environment (JRE), which is composed of a JVM and a collection of classes that implement the Java API for the underlying device operating system and hardware, runs alongside native applications that may be written in C/C++ or another language. The JRE can be one of many environments such as Java ME (Micro Edition) for embedded devices, Java SE (Standard Edition) for desktops and Java EE (Enterprise Edition) for server platforms. Each platform provides a different set of services which can be exploited by applications using the OSGi framework. The implementation of the MATCH framework discussed in this chapter uses the Java SE desktop JRE.

There are several options of OSGi Framework implementation; the MATCH framework uses the Knopflerfish V2 [135] implementation which has a number of desktop features to assist development (such as Eclipse plugins) but this could be replaced by any other implementation meeting the OSGi standard, such as Apache Felix [3] (standalone desktop), Equinox [64] (developed for the Eclipse project) and Concierge OSGi [69] (lightweight for mobile devices).

Bundles in OSGi are regular JAR (Java ARchive) files with an additional bundle.manifest file which specifies that bundle's dependencies (in terms of other packages required for operation) as well as which packages it provides to the OSGi framework which can be used by other bundles. The bundle manifest specifies an "Activator" class which is called when the bundle is loaded and unloaded from the framework. This Activator class is responsible for starting and stopping any services provided by the bundle as well as obtaining references to services it requires.

The MATCH framework is composed of 4 main subsystems - the Message Broker, Service Discovery, Task Manager and Interaction Manager plus a collection of application specific tasks and components. This is depicted in Figure 7.2. Each of these subsystems may be composed of one or more actual OSGi bundles depending on the function and complexity of the module. The left hand side of the figure represents some of the components which may be "outside" the MATCH framework - in this case each of the physical devices associated with interaction components are shown as such. It is possible for the framework to allow interaction with resources or native code external to the framework itself.

Each hardware device has an associated software bundle within the MATCH framework. These publish and/or subscribe to messages from the Message Broker subsystem. Each message is delivered using a *channel* which enables communication between two entities within the framework. These messages may be sensor traces, user actions, feedback to be delivered to the user etc. Messages can be rendered at different levels of abstraction - dumb devices may only publish voltage readings from an analogue source while devices with more inbuilt functionality may use messages at a higher level of abstraction (such as representing user actions).

Each component registers with the Service Discovery subsystem which assigns it a set of named channels which it can use to communicate on the Message Broker and acts as an index allowing components and their channels to be located. The Task Manager is capable of creating, starting, stopping and destroying task objects which are responsible for performing application level functions in order to achieve goals. Finally the Interaction Manager is capable of interfacing with the Service Discovery and Task Manager in order to *map* a task onto an appropriate input or output component using the approach described in

125
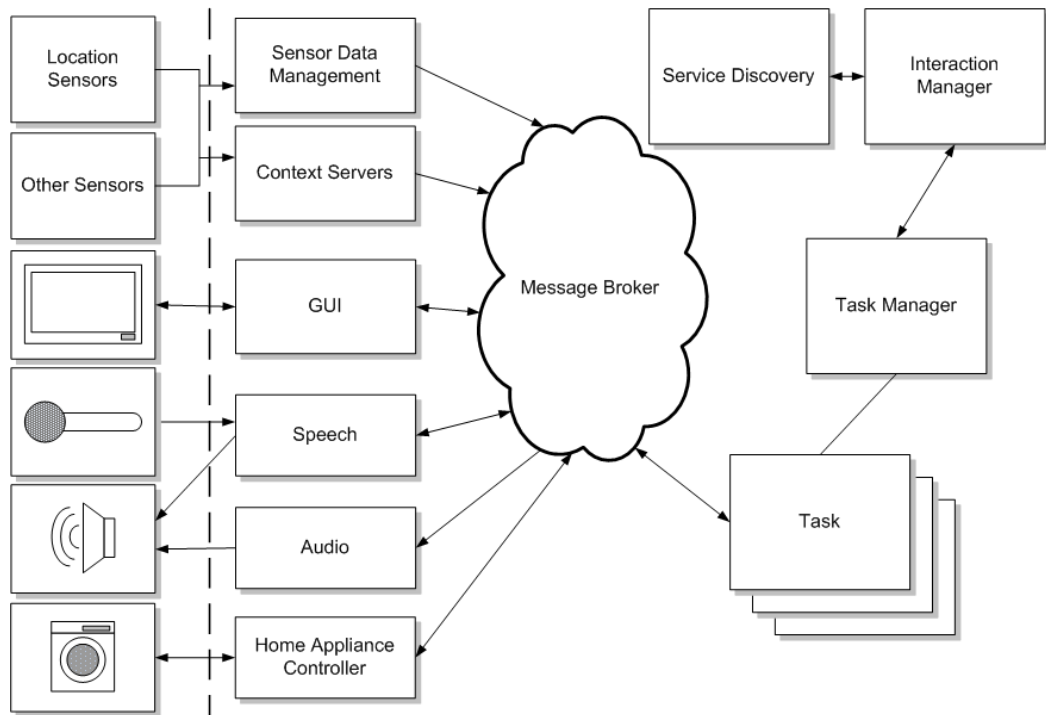
Figure 7.2: *High level MATCH Architecture Overview*

Chapter 5. Each of these major subsystems will be discussed in more detail in Section 7.2 and the Interaction Manager will be discussed even more fully in Section 7.3.

Figure 7.3 gives an overview of the MATCH architecture which will be used to present a worked example of the MATCH framework. In this example the goal is to alert the user if a desirable level of "busyness" is not maintained - this alert must be acknowledged by the user to indicate receipt.

A task (1) is created and started by the Task Manager (2) which is designed to accept messages of the type "Busyness" and output an appropriate notification if some required level of Busyness is not maintained. Additionally it is capable of accepting messages indicating the user's receipt of the message.

The Interaction Manager (3) is used to decide which components each of the tasks should map to. It does this by querying the Service Discovery subsystem (4) and by applying the model from Chapter 5. The Interaction Manager is able to determine appropriate interaction techniques based on the current context, i.e. where the user is physically located and what are appropriate ways of interaction with this particular user, by applying these criteria as additional evaluation functions. The choice of evaluation functions to use for this task was selected a priori by the user or system in this example. In this example, it is assumed the
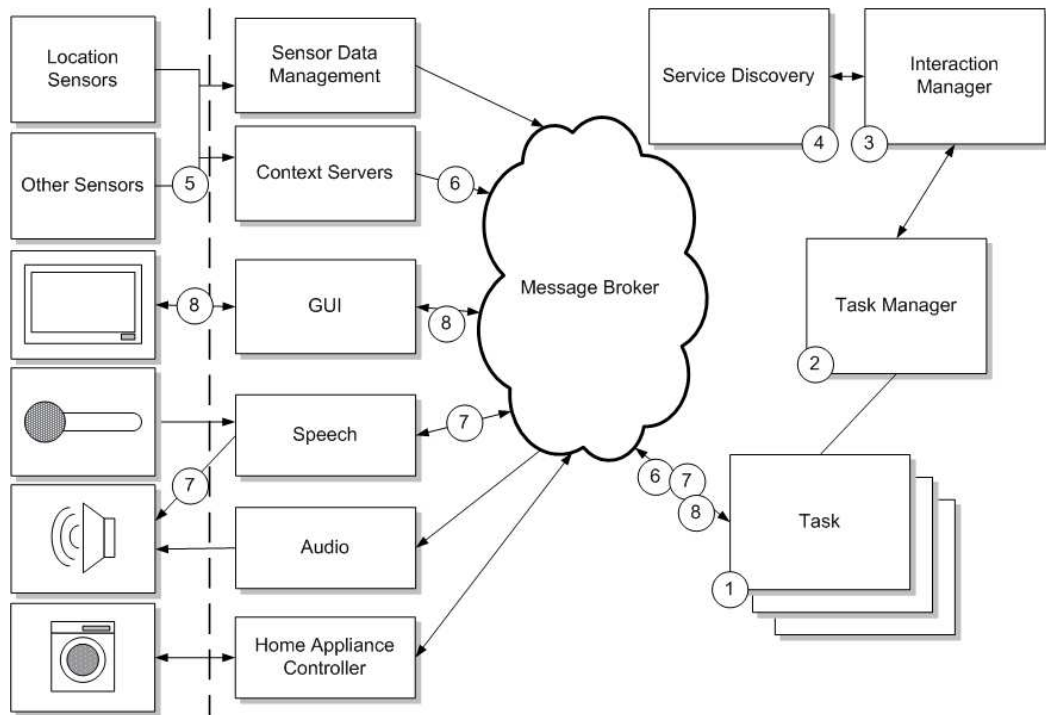
Figure 7.3: *High level MATCH Architecture Overview Walkthrough*

Interaction Manager has selected the Location and Door sensors via the Context Server for use in detecting Busyness and the Speech device for delivering the message and the GUI devices receiving notification of successful receipt of the message. Once the mapping is complete, the task can begin.

Sensor data, such as from movement or door sensors, is received by the Context server (5). The Context Server then interprets the sensor data to generate activity data (e.g. movement in bathroom) and publishes these messages, which are then received by the task (6) which has been bound to this sensor. This is facilitated by the Message Broker which operates as the main message distributor in the framework.

The task processes the busyness data and if necessary then sends an alert using its assigned modality (e.g. speech) (7) and waits on a reply. On receiving the message, the user will respond via a device (e.g. by pushing a button on the GUI). This will send a response back to the task, which then terminates (8). The task might have a time out waiting on a user reply. Hence if the user is inactive, and does not respond to the alert, the time-out will trigger a corrective action (e.g. contact house warden or a community nurse) which can be represented by an additional mapping to the device used to trigger the emergency alert and another iteration of this process.

This process has been simplified for clarity. For example, it is likely that to perform the mapping between task and component the Interaction Manager would be required to instantiate additional new tasks to perform data cleaning, processing or transformation (i.e. converting the alert into an appropriate Speech output) which has been left out from this initial summary but will be discussed later in this chapter.

7.2 Key Features / Subsystems

This section describes each of the key features within the MATCH framework. The following definitions are used in this section:

- Task - A task is a software component that can be started or stopped in the framework to provide some functionality and may be involved in possibilities. An example is a Speech synthesis task that converts text into audio.

- Application task - An Application task is a task that implements some application logic that aims to achieve some users high level goal. e.g. Notify me when the temperature gets too low.

- Component - A component, as defined within this chapter, is an endpoint in a possibility - usually representing a physical or software component/device which can communicate with the user.

- Service - In the context of the Service Discovery subsystem, a Service is a task or a component within the framework.

In this section the running example used in chapters 4 and 5 will be used to show how such a system would work in practise. Fred and Shirley are an older couple with chronic conditions that could be ameliorated by appropriate use of ubiquitous home care technology. Fred is hard of hearing and recently had a stroke and, although still physically fit, has become more and more forgetful since the stroke, requiring continual reminders for when to take his medication. Consider the medication reminder system for Fred previously discussed and shown previously in Figure 5.2 (Chapter 5).

7.2.1 Message Broker

The Message Broker uses an intermediary layer that transfers messages between components and tasks for decoupling. OSGi lacks native support for distribution of bundles across

multiple JVM's or physical machines but this is supported by using the Message Broker layer.

A key advantage of the Message Broker is that the transmitted messages are conceptually similar to the processes that are undertaken within a ubiquitous system. For instance, a bath temperature monitor sends regular updates on the temperature of the water which can be imagined as a message containing the integer value representing the new temperature.

The Message Broker is a publish/subscribe approach [72] where publishers of messages are not aware of the eventual destination of their messages. Instead, subscribers register to receive messages matching a programmatic pattern. In the case of the Message Broker, subscriptions are a content-based description of the types of messages they would like to receive.

Messages in the framework are composed of informally agreed data types and represented as a collection of key/value mappings. The type of the message is stored as one of the available keys. Messages contain a number of administrative details which are automatically appended by the Message Broker such as time/date, sequence number, priority and any other information which may be needed or used by the Message Broker as meta-data. Message types are identified by a string description similar to MIME types [81] which identifies the contents of the message; usually a fully qualified Java class. i.e. the Integer type might be identified as java.lang.Integer as the type field.
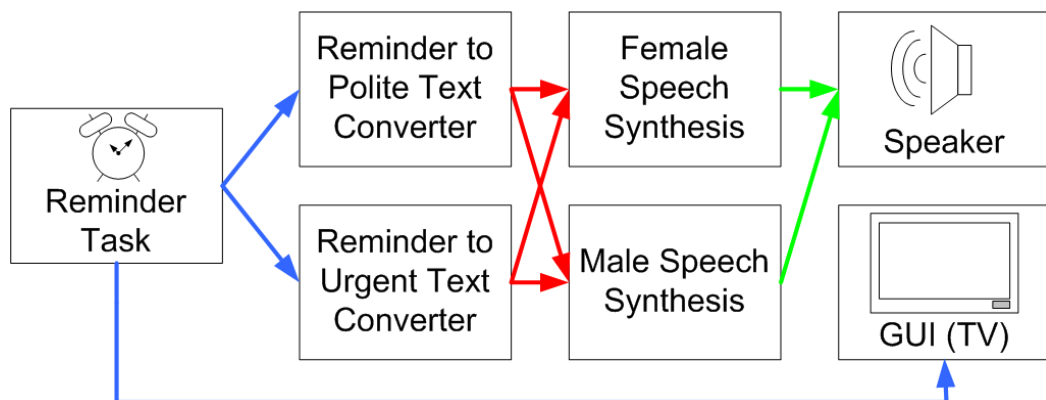
Figure 7.4: *Potential reminder message channels (Blue), textual message channels (Red) and audio message channels (Green)*

Figure 7.4 highlights the potential channels in Fred's reminder system. Types and transitions between types are indicated by colouring of the channel. Blue channels represent a high level reminder type while red channels indicate a transformed textual representation of the alert and finally green channels indicate an audio representation. Channels are more

specifically the points of communication with a task or component; the blue line connecting the reminder task and the GUI represents the connection between the *output* channel on the task and the *input* channel on the GUI.

The core of the Message Broker is the Router object which implements `subscribe`, `unsubscribe` and `publish` methods which can be called by components and tasks. The `subscribe` and `unsubscribe` methods accept an object of type Subscription which implements a single required method `matches` which executes upon a message returning true or false. The `subscribe` method further accepts a MessageListener object which is capable of receiving messages which match the subscription (that is the result of `Subscription.matches(message)` is true). Specialised subscriptions may be implemented by the listening task or components, and can execute arbitrary Java code provided within the matches method to match messages based on any arbitrary criteria, or may instead use a number of generic matching approaches.

Examples of generic subscriptions provided include SimpleSubscription which is a lightweight and fast subscription which is only capable of matching on message type and message channel and SubLangSubscription which is a middleweight implementation of a "subscription language". The subscription language implements a recursive descent parser [42] to evaluate a string expression of the form: `channel == "bathroomThermostat:temperature" && type == "Integer"` and is capable of evaluating a wide range of subscription requests.

In addition to type and direction, each channel has an associated channel identifier which can used as part of a subscription. If one component or task transmits on a channel then any other component or task which receives on its own channel with the same channel identifier can receive that message - thus connecting the two channels.

The Message Broker approach allows selection of appropriate messaging strategies based on the circumstances of the deployed application. Three different implementations of the Message Broker architecture have been implemented which are appropriate in different circumstances. The first strategy for the Message Broker is a "local only" Router which is implemented as a single Java object - storing subscriptions and processing messages entirely on the local machine. This is appropriate when performance is a factor but where it is not necessary to distribute the framework across multiple host devices. A second implementation uses the Elvin [201] protocol and results in a client/server configuration where multiple MATCH framework instances can communicate via a central server. Another implementation of the messaging architecture is based upon the REDS [47] protocol which creates a distributed peer to peer messaging system which itself offers a choice between reliable (TCP) and unreliable (UDP) messaging strategies. These brokers are implemented as OSGi bundles which can offer alternative messaging strategies between

and allows for their choice at run time rather than at development time.

The design of the messaging infrastructure provides a number of advantages. It helps to encourage loosely coupled applications based on agreed data formats while providing a explicit location to manage the behaviour of messaging such as threading, message priority and sequencing.

The Message Broker architecture is a general approach to the componentisation of sensor based systems. The publish/subscribe approach to interaction is not a necessary condition of the model presented in Chapter 5 but provides a convenient mechanism to allow interaction when components only have a minimal knowledge of the operating mechanics of other components. The Message Broker was provided to the OpenInterface project[1] who were interested in its use for multi-modal interaction.

7.2.2 Components

Components within the MATCH framework could be low-cost wired and wireless sensors strategically placed within a dwelling to collect data and might include under-carpet footfall sensors, passive infra-red (PIR) movement sensors and beam-break sensors to detect movement through doors as well as context sensors such as temperature, light level, power usage and vibration sensors. In addition to these sensors the MATCH framework can incorporate a multitude of user interaction devices such as graphical user interfaces, microphones and loudspeakers for audio interaction as well as independent devices such as mobile phones which can incorporate visual, audio and vibrotactile feedback.

These components may range from *dumb* sensors that can only report their current status to *intelligent* standalone devices which include a programming interface which allows them to be controlled from within the MATCH framework. As discussed in Section 7.1 the messages sent and received by components can be at different levels of abstraction while still being represented the same way. For example, some components may represent electrical level devices with no onboard logic while others may represent high level devices which are capable of representing user intention.

Figure 7.5 shows the two components in Fred's reminder system highlighted. In this example they are both output components which only have channels which receive messages.

All components within the framework are composed of unique identifiers and a collection of

---

[1]http://www.oi-project.org/
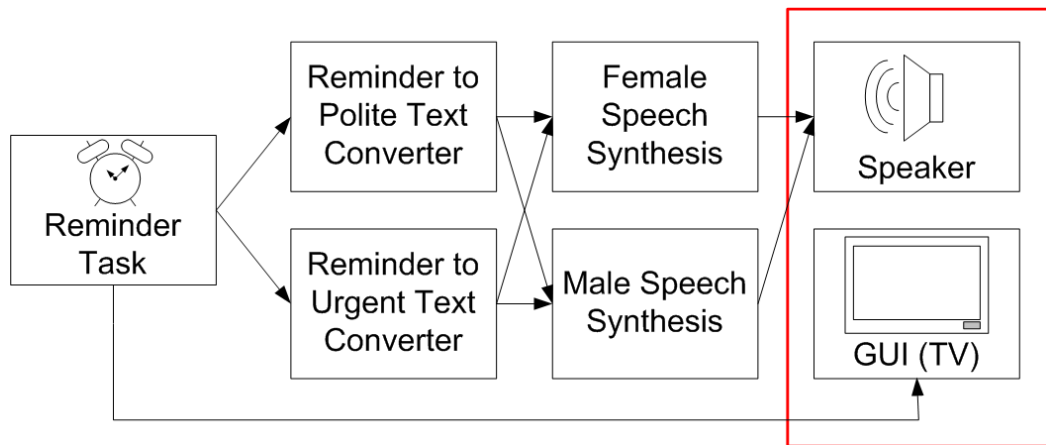
Figure 7.5: *Components*

one or more communication *channel* descriptions. Channels are associated with a specific component and contain a unique (to the component) name, a type and a direction - for example, the channel named "temperature reading" of a temperature probe component will uniquely identify that channel and it may have a the type Integer and be an Output channel (with respect to the component). Components can contain zero, one or more channels.

Components register themselves, along with a list of their channels (name, directions and type) with the Service Discovery subsystem described in the previous section. Components can obtain a reference to the Service Discovery system using OSGi service lookup functionality. Registration of the component with the Service Discovery system is simplified by the presence of an abstract class which can be extended by the component and which implements Service Discovery registration on behalf of the component.

A number of typical components implemented within the framework are described in sections 7.4.3 and 8.3.2.3.

7.2.3   Tasks

Another primitive within the framework is the notion of a task. Tasks are software components that implement system and user level functionality within the framework. Examples of this functionality might be the "Notify the user about an event" task or the "Convert a String into a Wave file using Speech Synthesis" task. Tasks can be both *Application Tasks* which are intended to support a user goal within the system - in the previous example the process of notifying a user about an event can be regarded as an application task which supports the users goal of being notified about an event. Tasks can

132

also be used to implement system functionality such as the Speech Synthesis task described above. This is a task that the user is not explicitly interested in but which can be used to support the users goal (by delivering the user notification via speech synthesis).

Application tasks would typically be started by a user (or a policy rule or similar previously set up by the user) to satisfy some need whilst supporting tasks would be started by the Interaction Manager, to be described in Section 7.3, in order to support an application task.
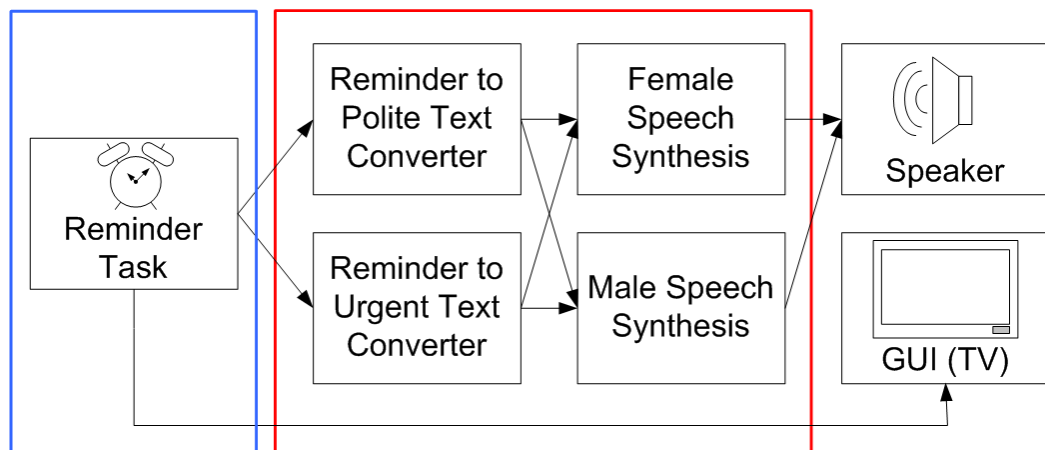
Figure 7.6: *Application tasks (Blue) and tasks within possibilities (Red)*

Figure 7.6 highlights the tasks within Fred's reminder example. The blue box contains an application task (the reminder task) which contains application level logic; this logic is responsible for deciding when to issue a reminder and what the content of the reminder should be. The tasks highlighted in the red box are available to be used as part of a possibility; these tasks perform some function or implement a required ability that is necessary to transform a reminder into an audio alert which can be delivered to the Speaker component.

Tasks can be standalone single tasks or may contain other subtasks modelled as a directed graph composed of other tasks, similar to Concur Task Trees [169]. To facilitate this; tasks can either be a concrete task implementation (Java Code) or a task structure description (XML description of subtasks) which is used to describe how tasks are combined together. Tasks are identified by URI consisting of a scheme (i.e. http) and a scheme specific part (i.e. www.example.com).

Concrete tasks (with a scheme of "class") which consist of Java classes (identified by the scheme specific part of the URI) can be dynamically created using Java reflection capabilities. All other schemes are considered to refer to a higher level XML task

description which allows for the composition of subtasks and are loaded by a scheme specific resolution to access the XML document that describes the task structure. In addition to the standard schemes provided by Java IO mechanisms (local files, http, ftp etc) there is an additional *taskrepository* scheme which allows temporary task descriptions to be stored in memory within the framework - this can be used for user specified task descriptions not yet saved to disk.

Tasks, like components, contain a collection of communication channels and are parameterised by message type and direction. Tasks can be instantiated multiple times (for example multiple notification tasks) and can accept parameters to create more specific instances of a generic task. When the classes used for a task are loaded into the framework (but before the task is actually started) a *task description* is registered with the Service Discovery; this task description contains all the necessary information (information about the channels a task has, the task URI) necessary to instantiate an instance of a task.

Unlike components with fixed channels, a tasks channel can be reconfigured such that the task can change the channels it is using. The selection, and binding, of a task's channels is handled by the Interaction Manager - discussed in detail in Section 7.3. Tasks are assigned a *channel mapping* proxy object for each of its channels and it uses this object to publish or subscribe to the channel. The channel mapping proxy also notifies the task when any changes occur to that channel which allows the task to respond to configuration changes if necessary.

The Task Manager is responsible for instantiating, maintaining and destroying tasks within the framework. Given a task description, the Task Manager will create an appropriate task (or collection of subtasks) and register a task object within the Service Discovery system to indicate the existence of the new task instance. The Task Manager is responsible for starting, stopping, pausing and resuming tasks (tasks can exist but not be running). Each task exists within a dedicated thread for its lifecycle.

7.2.4   Service Discovery

In order to allow components and tasks to communicate directly with each other via the message broker, a mechanism is required for discovering which other services are available; this is accomplished by the Service Discovery subsystem. Like the Message Broker, there could be multiple variant implementations of the subsystem.

There are many Service Discovery protocols which allow the automatic discovery of services and devices; including Jini [6], Simple Service Discovery Protocol (SSDP) [93],

Service Location Protocol (SLP) [225] and Universal Description Discovery and Integration (UDDI) [160].

There are two main approaches to Service Discovery which can be broadly categorised as *broadcast queries* and *service registration* approaches. Broadcast query based approaches (such as SLP and SSDP) require all devices offering services to listen on a well known port (often multicast UDP) for queries from other devices for service discovery requests that it meets the specification for. When a service responds to a request it can either respond directly to the interrogating device (using information in the original request) or multicast its reply back using the same approach it received the request. This approach requires no service discovery specific infrastructure.

The service registration approach (such as Jini and UDDI) requires services to register their available functions in a database service which allows queries to be made to find registered services. When a service is needed, the interrogating device queries the database rather than attempting the find the service itself. This approach requires additional infrastructure but can result in faster queries, as not every service needs to respond, and simpler implementation, due to no need to develop a network protocol. However, this does require that services are able to update the database whenever the service status changes (such as becoming unavailable at a later point in time).

The approach used in this framework is to define a Service Discovery interface, which could conceivably be implemented using either approach. However, for simplicity and ease of implementation the framework was used entirely with service registration based implementations.

The Service Discovery subsystem implemented in the framework must allow other services to obtain lists of components, tasks and the message types that can be used to interact with them. Components, tasks and message types are registered with the Service Discovery at the point of first usage. A reference to the newly added object and its relationships to other existing objects is then stored with an internal mapping structure. This allows rapid queries to determine the available services.

The Service Discovery subsystem is capable of notifying other services when services are added or removed from the framework. Other subsystems or services can act in response to these changes and reconfigure themselves appropriately - services can register observer/listener objects which are notified by the Service Discovery subsystem when a service status changes.

There are clearly many approaches to implementing a Service Discovery subsystem. The approach taken here is based on the service registration technique used by systems such

as Jini and UDDI as discussed previously. In this model, each service is represented by a description of its available channels and properties.

Services wishing to register themselves with the Service Discovery instantiate an instance of an appropriate description class (task description, component description etc) with a unique identifier. Other concepts within the framework (Such as channels and message types) are similarly described by description class which can be referred to by other entries within the Service Discovery.
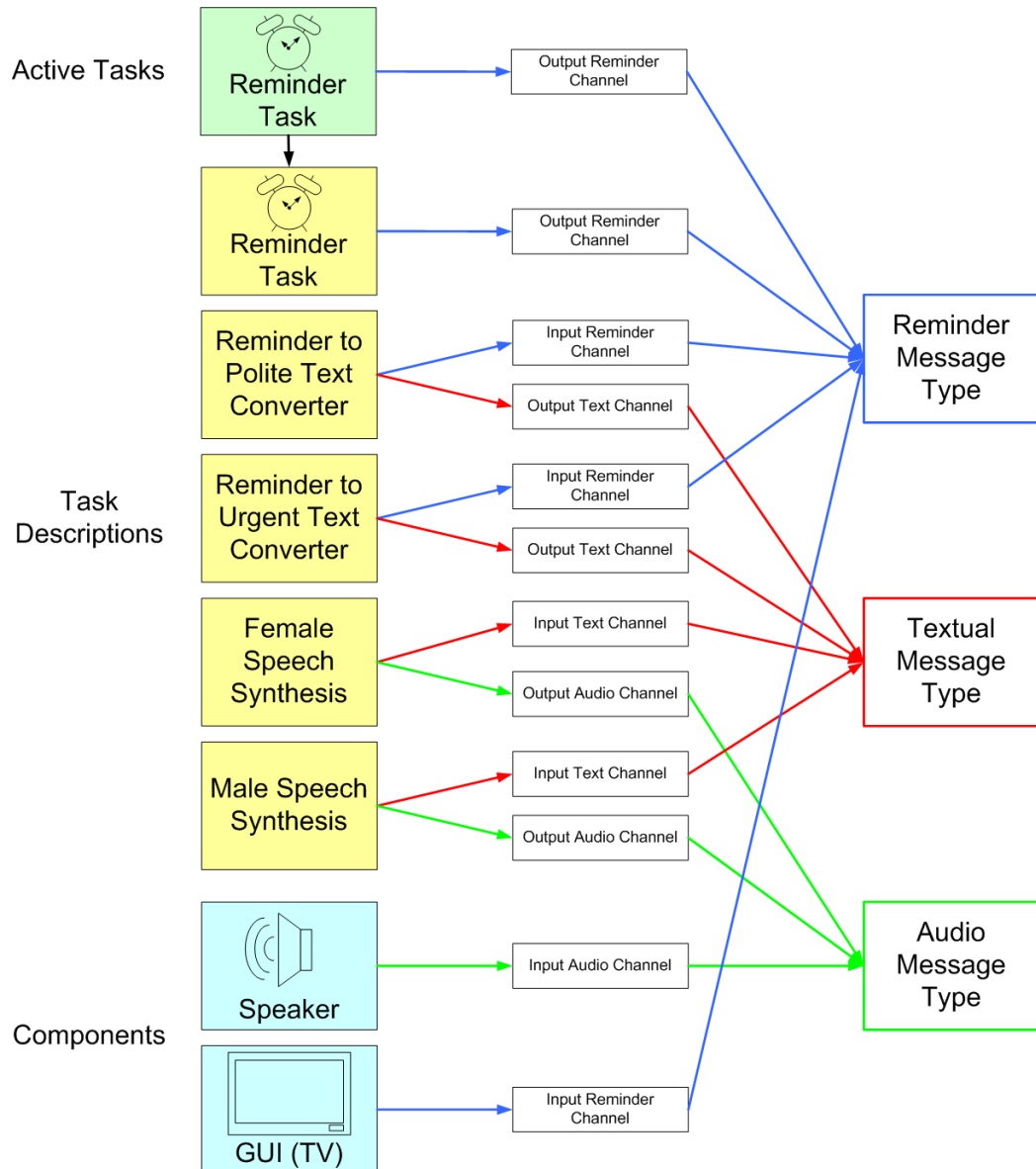
Figure 7.7: *Service Discovery references*

136

Figure 7.7 provides an example of how Fred's reminder system could be represented inside the Service Discovery subsystem where the directionality of the arrows indicates a reference from one entity to another and is not indicative of message data flow. The leftmost column contains active tasks, task descriptions and components while the centre column contains a list of channels belonging to the other entities and the rightmost column contains the descriptions of message types channels can have. References between channels and types are coloured, as in Figure 7.4, to aid readability.

Notice in particular that the reminder task is listed twice; one of these entries is a task description (yellow) which describes the task and contains the information (class names, required parameters etc) necessary to start the task while the other represents an actual running instance of the task (green), of which there may be many. The active task can contain a reference to its task description.

Not shown in this example is the capability for tasks to be formed into hierarchies (for example, the two Speech Synthesis tasks could be children of a single parent Speech Synthesis task) and for message types to indicate subclassing (i.e. ability to cast from one message to another). Additionally, there is an alternative Ontology based implementation of the Service Discovery subsystem which is discussed in Section 7.4.3.7.

7.3   Interaction Manager

The Interaction Manager is responsible for managing the interaction portions of the task such as deciding which devices a task should use for communication and establishing the mapping between a task and selected components.

The Interaction Manager is primarily responsible for determining which components can be used by a task, and then determining the best component to use from the set of candidate components. The Interaction Manager is an implementation of the model described in Chapter 5; the process has been previously described but is summarised below.

The first step is to create a graph representing the current system by obtaining a list of components and tasks from the service discovery subsystem. This list can be transformed into a directed graph with each component and task represented as a node. Compatible components and tasks are connected with edges via compatible message types. Finally the application task, that the evaluation function has been assigned to, is added as the root node and connected to a compatible message type via the channel that is being configured. This graph can then be used as the basis for evaluation for the selected application task.

A graph traversal is run to determine the set of possibilities that are available to use. This set of possibilities represents each of the different ways that it is possible to configure this task.

This set of possibilities can then be examined using evaluation functions to measure various aspects of each possibility, ranging from user preference and system resources to contextual factors such as location and noise level.

Once evaluation functions have been executed, the possibilities evaluated, and the choice made, the Interaction Manager is responsible for implementing the possibility by starting any required tasks and connecting them appropriately.

In this section a notion for Method signatures, such as Method 1, derived from the Java Language Specification [96] is used in order to elucidate the structure through more concrete examples.

7.3.1 Preparation

The construction of an Interaction Manager which is capable of implementing the required techniques will now be discussed. To do this, a walkthrough of the Interaction Manager evaluation process will be presented and the actions that take place at each step described.

The Interaction Manager is assigned to evaluate an evaluation function for a specific task and a specific channel belonging to that task. For example, consider an alerting task with two output channels; one channel dedicated to two different people. Each channel needs to be evaluated separately should you wish to deliver the information to different people. As such, through this section, the Interaction Manager will be described as evaluating for a channel on a specified task. References to the application task should be taken as referring to this combination of task and channel. Note that a single evaluation may not represent the entirety of a configuration in which a task is involved; the other channels of a task may (and often will in cases when a task receives input from a user, performs some process and renders an output) be assigned with different evaluation functions.

The Interaction Manager maintains record keeping using a selection of map structures, which are indexed by a combination of the task and the name of the channel on the task, on which an evaluation function has been set. The Interaction Manager is notified of a request to assign a new evaluation function to a named channel on a task. This occurs by the calling of the `setEvaluationFunction` method shown in Method 1.

Method Signature 1. *void setEvaluationFunction(Task, ChannelName, ApprovalEvaluationFunction)*

The purpose of this method is to assign an evaluation function to a channel on a task. There are a number of subtle conditions which should be considered as part of this method. If this evaluation function is already the assigned evaluation function for this channel on the specified task then evaluation proceeds no further and immediately returns from this method. This is to prevent unnecessary re-evaluation of the evaluation function as will be shown in a moment.

The concept of stimulus-based re-evaluation is discussed in Section 6.6.3 where the idea of a re-evaluation listener, added to a child evaluation function, is introduced. At the root of the evaluation function tree it is necessary for the Interaction Manager to add the same listener to the evaluation function which it has been assigned such that when the listener is triggered the evaluation function can be reevaluated. It is necessary to check that the evaluation function has not been seen before to prevent adding multiple listeners to a single evaluation function.

If an evaluation function has any initiation methods (for example to create or connect to data stores) these are called and assignment of the evaluation function to a task and channel is stored.

At this point it is not yet required to invoke the evaluation function. Until a task has requested a ChannelMapping object it is not necessary to create a mapping at this point and may be detrimental to do so as it may involve allocation to devices which could otherwise be used by other tasks (in the case of components which can only be used by one task at a time).

If there was a previous evaluation function assigned it is, however, necessary to perform an evaluation to allow the new function to be executed and the configuration updated. Otherwise, the evaluation is not conducted until the task requests. For this example, assume that the task has now requested a ChannelMapping object to which it can publish messages.

A task will request a ChannelMapping object by calling the `mapChannel` method of the Interaction Manager (Method 2). This is responsible for returning a ChannelMapping object. ChannelMapping objects are stored by the Interaction Manager in a mapping / dictionary structure using the task and channel as the key and the assigned ChannelMapping object as the value.

Method Signature 2. *ChannelMapping mapChannel(Task, ChannelName)*

If a task already has a ChannelMapping object then it is not necessary to reevaluate the evaluation function as this only needs to be done when the evaluation function itself signals that a re-evaluation is necessary - in this case it is possible to immediately return the existing ChannelMapping which was previously stored.

Otherwise, it is necessary to execute the process described in Section 5. This is done by calling the Interaction Manager's `evaluateMapping` function as shown in Method 3 which will actually perform the process. This method can be threaded or parallelised if necessary to allow multiple evaluations to occur concurrently but the process will be explained for a singly threaded implementation.

Method Signature 3. *ChannelMapping evaluateMapping(Task, ChannelName)*

Earlier in this section it was explained that if a previous evaluation function was present when an evaluation function is set then the evaluation function can be immediately reevaluated - this would be done by calling the `evaluateMapping` method.

7.3.2 Building the Graph

The first step of evaluating the configuration request is to retrieve a list of available devices and build a graph which can be used to determine the set of available possibilities using the buildGraph method shown in Method 4.

Method Signature 4. *Graph buildGraph(Task, ChannelName)*

Each of the nodes within the graph is an entity within the Service Discovery system and directional edges are used to indicate an ability to traverse the graph in that direction. For instance it is possible to go from String messages to Wave file messages via a Speech Synthesiser task but it is unlikely to be able to do the reverse using the same task.

In this implementation, a graph is defined as being a collection of nodes, each of which is mapped to a set of edge objects which contain references to the two nodes that this edge connects and the direction. If implemented as a hashtable this provides O(1) access to lists of edges for a node, using a hashset implementation of the list of edges additionally allows O(1) addition and membership tests.

A root node is specified for the graph and is set equal to the instance of the node which represents the application task.

To build this graph designate a single node which represents the root node - i.e. the

task/channel pair that the evaluation is being performed for. Retrieve collections of the available channel types, components, active tasks and available tasks from the Service Discovery.

Iterate through the list of channel types and add each of them to the graph then create a directional edge between the root nodes and the channel type it is associated with (the type of messages that it can send/receive).

If the underlying architecture supports casting from on message type to a super type then this can be represented in the graph. Automatic casting for specific types to the more general type can only be conducted in one direction as, for example, it is always possible to ascertain that an Integer is a Number but not that a Number is an Integer. Iterate through each channel type and add an edge between that channel type and its supertype. Note that there is no need to include edges to all of its supertypes - only its immediate parents.

Add a node for each component, active task and potential task; add nodes for each of the channels on the components and connect them to their component. Add edges between each of the newly created channels and their channel type node. For edges between the newly added component and channel, and between the channel and its channel type it is important to maintain the correct direction of the edges to match the direction of the channel such that when these edges are followed that the elements are not connected backwards.
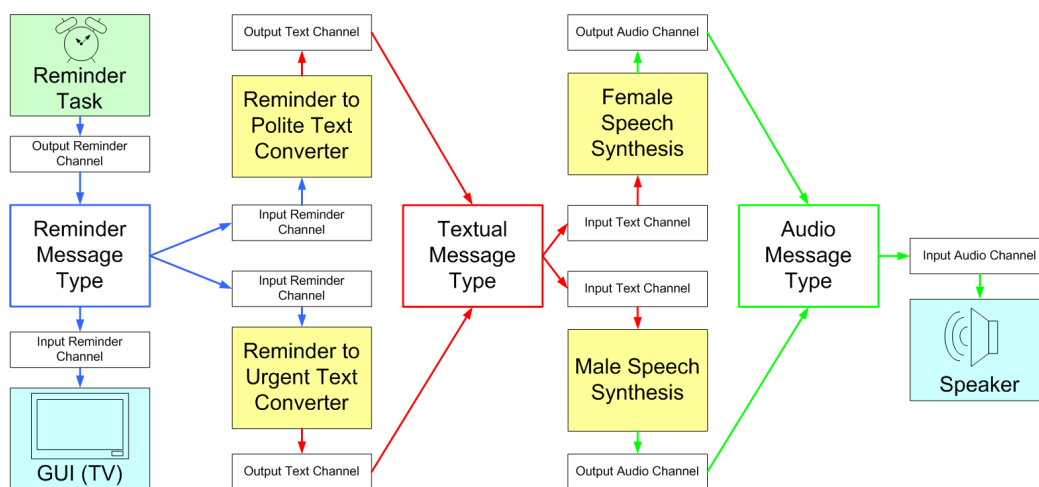
Figure 7.8: *Detailed Graph*

Figure 7.8, coloured as previously, shows a detailed form of the graph previously shown in Figure 5.2. This graph represents the graph building that results from the algorithm described above given the Service Discovery configuration described previously. This particular implementation of the graph building differs slightly from the generalised form

141

described in Chapter 5 in that it explicitly includes both message types and channels as nodes within the graph although note that the structure of the graph is essentially the same. It was chosen to include both message types and channels as nodes within the graph as this simplifies the logic required to traverse the graph (all entities within the graph are homogeneous for traversal).

7.3.3 Generating Possibilities

Once an efficient graph has been created to represent the available services it is possible to traverse this graph in order to build a list of the valid possibilities that can be chosen - this is performed by the `getPossibilities` method which accepts the previously generated graph as its input as well as a starting Task and Channel which the graph is traversed from, as shown in Method 5.

Method Signature 5. *List<Possibility> getPossibilities(PossibilityGraph, Task, ChannelName)*

To perform this traversal a recursive method `traverse`, shown in Method 6, is used which implements a variation on the Depth First Search algorithm in order to execute a traversal and obtain all possibilities.

Method Signature 6. *List<Possibility> traverse(PossibilityGraph, CurrentNode, List<VisitedNodes>)*

This method accepts a graph, a current node and a set of already visited nodes which should be avoided during the traversal. Initially this is called by *getPossibilities* with the graph to be traversed, the root node and an empty hashset of visited nodes; using a hashset allows for quick checks to determine if an edge should be followed.

The `traverse` method initially creates a list of possibilities; initially containing only the node it was called with (*currentNode*) and adding itself to the set of visited nodes. It then retrieves a list of all edges for the current node which have not been visited and recursively calls itself using the end node for each edge as the new *currentNode* and creating a shallow copy of the *visitedNodes* set for each recursive call.

Each of the returned possibilities has an entry for this *currentNode* prepended to the possibility to represent the fact that these possibilities are only accessible through the current node.

The set of possibilities, the *currentNode* and all recursively generated possibilities with

currentNode are prepended, will then be returned to the caller. In the case of a node without unvisited edges then this returned list will only consist of a reference to itself and acts to terminate the recursion. Intermediate nodes within the traversal (for example one removed from a terminator node) will return a list of all possibilities which can be reached through this node while excluding the possibilities that have been generated by its parent caller. The root node will therefore return a list of all reachable possibilities using this node as a base. Unreachable nodes, whether by disconnection from the graph or impossible to traverse edge directions, will not be represented.
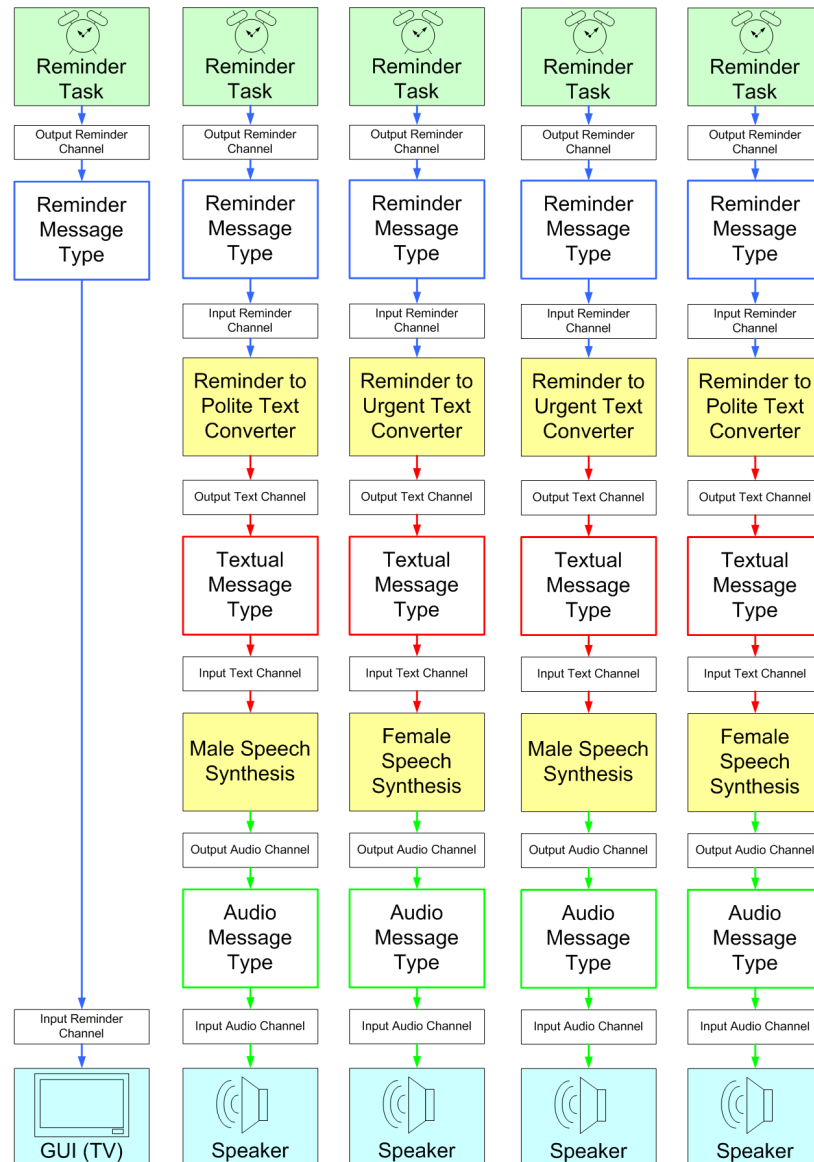
Figure 7.9: *Valid Possibilities*

Figure 7.9 shows the valid possibilities that would result from such a traversal of the graph. Note that this does not show a selection of possibilities which would be regarded as being *invalid* using the graph traversal described above. Specifically such invalid possibilities would primarily consist of segments of possibilities which did not terminate in a component. The next section explains how to deal with these.

The traversal function described here has a small number of limitations. Specifically, the Depth First Search approach does not take into account during the traversal loops composed of idempotent tasks. This poses the question of to what extent loops should be evaluated during the traversal - traversing a graph with a loop indefinitely would result in an evaluation which would never terminate; in this implementation the design decision was taken not to follow loops at all which has the benefit of reducing the complexity of the algorithm whilst accepting the drawback that some subset of valid possibilities may be ignored. Alternate traversal algorithms can be used which will follow a loop to a fixed degree or until some heuristic is met, alternately improvements to the traversal algorithm are discussed in Section 9.2 where traversal of the graph can be directed heuristically.

7.3.4 Evaluating Possibilities

In order to evaluate the possibilities, the evaluation function is first retrieved from the mapping of function to task described previously. If no evaluation function has been set for this task and channel there are two options; (i) choose a "default" evaluation function - for example one that rejects all possibilities and maps to nothing, or (ii) raise an exception and abort the evaluation process. The first approach is used in this implementation and an evaluation function which always returns an empty set of possibilities is used when no other evaluation function is provided.

A list of all potential possibilities was obtained in the previous section. However in the case of some system implementations, these may not all be valid possibilities. For example, in the MATCH framework, it only makes sense to instantiate possibilities that terminate in a component or task; possibilities that terminate in a channel type node would not be considered as being sensible possibilities.

Approval evaluation functions can be used which perform sanitizing operations upon the lists of possibilities - an example would be an endpoint checking evaluation function which approves only possibilities which contain a component or task as the terminator in the possibility. Additional checks or restrictions can be performed at this stage to restrict the available possibilities further if so desired; for example, restricting particular devices only

to specific users to prevent a user supplied evaluation function from selecting disallowed devices.

Now, consider the specification for evaluation functions introduced in Chapter 5. When a single evaluation function is being used, that function accepts a set of possibilities which has been generated by the previous stage in the model, performs some process, choice or algorithm upon them, and return a set of possibilities which should be instantiated. The method signature for such a function is shown in Method 7.

Method Signature 7. *Set<Possibility> evaluate(Set<Possibility>)*

The Java practice of specifying the type of a collection in brackets has been adopted. Functions that implement a method signature as shown in Method 7 are classified as belonging to the broad category of *approval functions*. Approval functions are designed to filter a set of possibilities and return a subset of possibilities.

Instead of returning a list of possibilities an evaluation function could instead return a metric to sort them into some order. An example of such a signature is shown in Method 8.

Method Signature 8. *Map<Possibility,Score> rank(Set<Possibility>)*

Method 8 accepts a set of possibilities and generates a metric for each of them that it returns as a mapping from possibility to the metric. The metric might be a simple integer or may be a more complicated object representing upper and lower bounds for the applicable metric. Metrics may be scalar (absolute ranks), ordinal (relative ranks) or nominal (categorical).

Additional examples of ranking possibilities might include the return of an ordered list of possibilities instead of the unordered set that it was initially given, as shown in Method 9. Other possible method signatures for more complicated evaluation function results can include subdividing possibilities into sets of possibilities, as in Method 10, or relationships between possibilities to determine similarity or relationship between them as in Method 11.

Method Signature 9. *List<Possibility> rank(Set<Possibility>)*

Method Signature 10. *Set<Set<Possibility>> subdivide(Set<Possibility>)*

Method Signature 11. *Relationship<Possibility,Possibility,Score> relationships(Set<Possibility>)*

These different evaluation function method signatures deliver different types of results to the calling process. As such it is important to note that evaluation functions are not limited

to a single method signature but can actually accommodate a variety of input parameters and return values.

At first, it seems advantageous to allow all evaluation functions to be called directly by the Interaction Manager but this approach would require that the Interaction Manager be able to deal with all possible method signatures that might be encountered. The centralisation of this knowledge is undesirable as it would be preferable to be add in new types of evaluation function at runtime without needing to redeploy a new instance of the Interaction Manager. Instead, a modular approach to the problem is taken which works by informal data type agreement where evaluation functions can implement a variety of different calling syntaxes. The Interaction Manager is restricted to understanding a single calling syntax as in Method 7. Evaluation functions can however call other evaluation functions with a different calling syntax to their own - this allows for an extensible approach both to adding new approaches of combining evaluation functions as well as combining different approaches within a single evaluation tree.

As a result of this decision the Interaction Manager only needs to be aware of the method signature described in Method 7 which limits the required complexity of the calling process as it only needs to concern itself with a single type of evaluation function and associated method signature. The remainder of this section will show how this can be realised.

As previously discussed in Chapter 5, the evaluation function combination structure is modelled as tree. To achieve the decoupling described above, the root of the tree is a function with the approval method signature, called to determine the results of the evaluation for the entire tree. In order to do this, the root function delegates to functions directly below it in the tree, which can in turn further delegate to functions below them. In this approach the tree is defined as rooted at a particular function with each edge to another evaluation function being represented by that function's membership within a set of evaluation functions for that function.

If the Interaction Manager is only capable of calling approval method signatures but the functionality that it desires is located in some evaluation function which implements the ranked method signature (i.e. this could be a COMET-style context sensitive function returning a ranked list of appropriate possibilities) but not the approval method, then an *adapter* evaluation function is required to allow them to be combined together.

Such an adapter implements the approval method signature but is capable of calling other evaluation functions which have the ranked signature and performing some operation on the result before returning it to the original caller as shown in Figure 7.10. Note that the Approval Evaluation Function can not maintain the same semantics of the ranking

function as it returns an unordered set - the operation performed for a ranked to approval adapter might be some thresholding (any possibilities where the rank was greater than some threshold *n*), selective (top *n* results) or complete (all results).
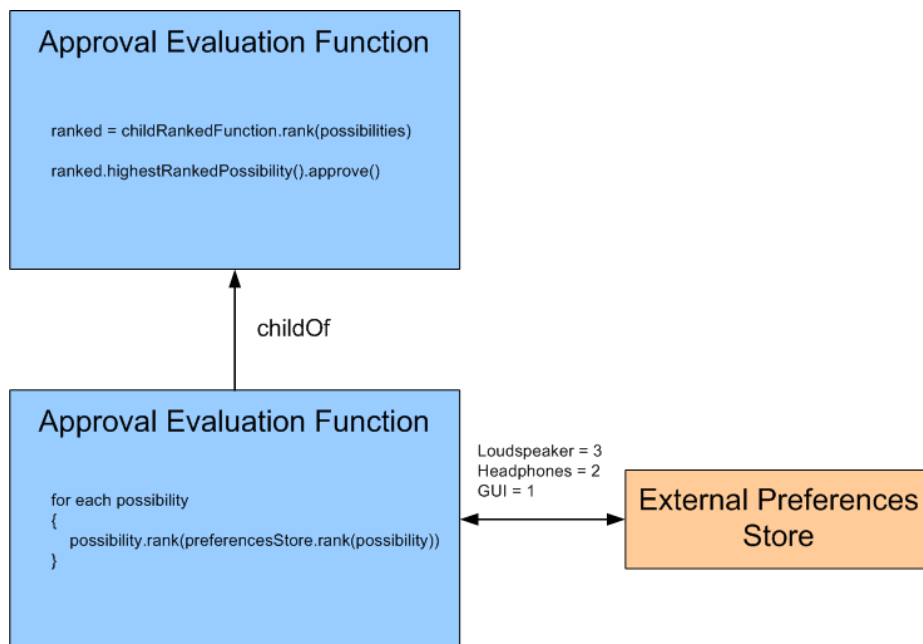
Figure 7.10: *An approval evaluation function acts as an adapter for a ranked evaluation function*

These adapter evaluation functions can be used to allow for evaluation functions with a completely different method signature to the one the calling process expects to be used within the tree. Any given evaluation function may implement multiple signatures; for example the context sensitive function could have signatures for both approval and ranked behaviour.

Using this approach, each evaluation function accepts only selected types of evaluation function as its children (those implementing the correct form of the evaluate function) and can call the child function's evaluation function directly. This allows combination of disparate evaluation functions and different approaches to combining functions, such as those in Section 6.5.4, to be used simultaneously within a single evaluation function tree.

The Interaction Manager need only concern itself with those evaluation functions which meet the method signature of an approval function as previously shown in Method 7. That is, one which accepts a list of possibilities and returns a filtered (approved) list of possibilities. The evaluation function (or a tree of evaluation functions with an approval function at the root) is executed at this stage by calling the `evaluate` method with

147

the list of possibilities and receiving the approved list in return. Control passes to the evaluation function itself which analyses the possibilities and ultimately returns a listing of possibilities which have been selected.

7.3.5 Implementing Possibilities

Given the result from an evaluation function of a list of possibilities, the semantics of the Interaction Manager is that it will attempt to instantiate all of the possibilities that have been returned to it by the root evaluation function. This process involves starting these possibilities, storing a reference to them to stop them later if necessary and returning or updating appropriate ChannelMapping objects.

If there has previously been an evaluation function used for this task and channel, or this evaluation function is being reevaluated due to a change in circumstances, then the possibilities that have previously been started and are no longer required should be stopped. However, any possibilities which are already in use and which are in the set of possibilities approved by the evaluation of the current evaluation function should not be stopped. The `updatePossibilities` method shown in Method 12 is responsible for starting and stopping possibilities using this logic in order to obtain the new desired state.

Method Signature 12. *void updatePossibilities(Task, ChannelName, List<Possibilities>)*

A list of existing possibilities from previous evaluations can be used alongside the list of approved possibilities to decide which possibilities to start and which to stop. To do this a list of possibilities to start is created and is initially set equal to the contents of the approved possibilities; any possibilities found in the existing set of running possibilities are removed. Similarly, the list of possibilities to stop is initially set to be equal to the set of existing possibilities and any possibilities in the approved set removed.

More formally, the set of possibilities actually started is the difference between the set of approved possibilities and the set of existing possibilities, while the possibilities to be stopped is the difference between the set of existing possibilities and the set of approved possibilities. The union of these sets constitutes the possibilities which are both desired and are already being used and, as such, do not require changing.

The process of starting an individual possibility is delegated to the `startPossibility` method shown in Method 13 which is executed for each possibility in the list of possibilities to be started. A similar method exists for stopping possibilities, which is not discussed here, but which operates by reversing the actions taken in `startPossibility`.

148

Method Signature 13. *void startPossibility(Task, Channelname, Possibility)*

In order to start a possibility within the MATCH framework it is necessary to create the task objects and bind the channels together such that they can communicate. Using the Message Broker architecture this only requires that the channels share a channel identifier.

An efficient algorithm to do involves a sweep through the possibility starting from the end of the possibility (the fixed component that the task is being connected to). A single variable is used to note the channel identifier of the *current* channel. When a fixed channel node is found (such as the fixed channel on the destination component) then the current channel identifier is set equal to this value. When a task that needs to be started is encountered then it is started via the Java reflection capabilities and the current channel identifier is set to a new globally unique id (GUID). Any unset channels that are encountered are set equal to the current channel identifier at the time they are encountered. As a result, when two channels are encountered immediately after each other they will inherit the same channel identifier which causes them to be connected.

Figure 7.11 shows this algorithm being executed on one possibility on the left, and the resulting assignment of channel identifiers on the right.

The above approach works when only one possibility needs to be started as the result of an evaluation. However, if multiple possibilities are needed then the final stage of assigning a channel identifier to the application task's channel will fail as multiple identifiers will be assigned to a single channel. There are several possible solutions to this: (i) make it such that the ChannelMapping objects are capable of subscribing and publishing to multiple channels, or (ii) use proxy objects which connect the application task channel and the final channel in the possibility but which have no effect on the messages otherwise. The latter approach was taken in this implementation and "forwarding tasks" were used to allow multiple possibilities to be assigned to one task.

This thesis does not claim any contribution in terms of correct behaviour for transition and continuity between one configuration state to another as this work is focused on the choices of appropriate configuration states. For research which attempts to address these problems the reader is directed to the work of Florins et al. in graceful degradation which aims to guarantee maximal continuity between configuration states (typically of user interfaces) and in maintaining that continuity across changes [79, 80]. Specifically this thesis does not address the process of ensuring the the transition between configurations maximises continuity, although future work could be to design evaluation functions which minimise loss of continuity.
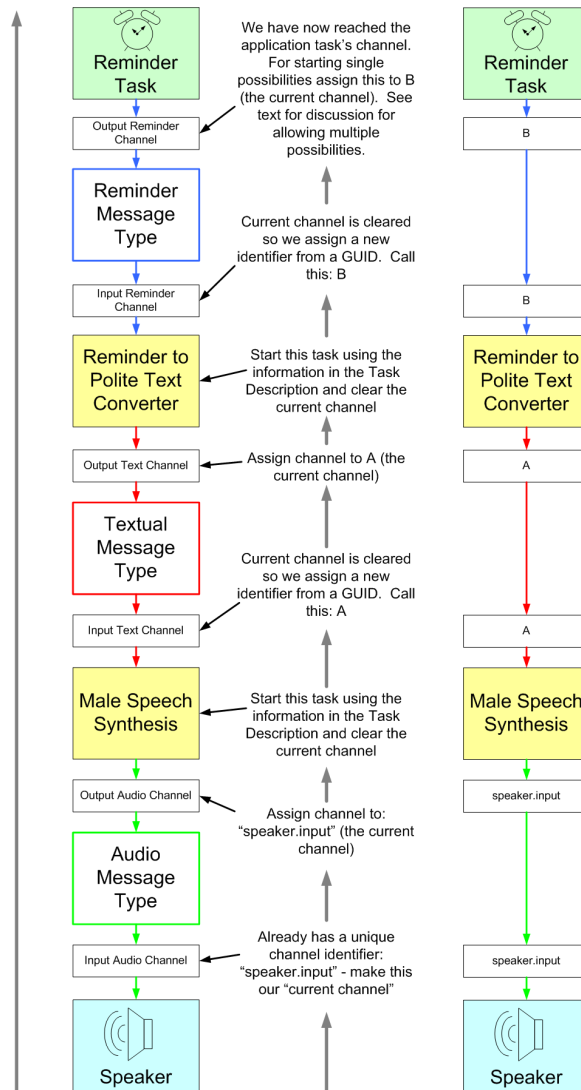
Figure 7.11: *Implementing a Possibility*

7.4 Implementation Validation

Validating infrastructure middleware and frameworks which are designed to support the implementation of other applications is a challenging practise. Edwards et al. [66] provides a number of guiding lessons learned from experience in evaluating these types of framework in the past which are briefly summarised below;

- **Prioritise Core Features** - Test a minimal set of core features early.
- **Build prototypes with high fidelity for expressing the main objectives**

of the middleware - The new features introduced by the framework should be the ones first tested.

- **Any test-application built to demonstrate the middleware must also satisfy the usual criteria of usability and usefulness** - The more ambitious the application the more this is true.

- **Initial proof-of-concept applications should be lightweight** - Testing should not require investment in a myriad of features to create a coherent real-world application.

- **Be clear about that your test-application prototypes will tell you about your middleware** - Every prototype should demonstrate something that has not been demonstrated with previous prototypes.

- **Do not confuse the design and testing of experimental middleware with the provision of an infrastructure for other experimental application developers** - It is hard enough to build your own experimental applications without supporting all possible features and extensions for external developers who may request interface changes which break another developers live application. APIs should be stable before inviting other developers to build upon the framework.

- **Be sure to define a limited scope for test applications and permissible uses of the middleware** - The correct methods of using and taking advantage of the middleware should be made clear to developers.

- **There is no point in faking components and data if you intend to test for user experience benefits** - Simulations of devices or features risk hiding interesting observations about how users actually use the applications.

- **Understand that the scenarios you use for evaluation may not reflect how the technology will ultimately be used** - Evaluations are for investigating the usefulness of the middleware - not in determining marketability or sales potential for a specific application.

- **Anticipate the consequences of the tradeoff between building useful/usable applications versus applications that test the core features of the middleware** - The best applications to test the core features may not be the best applications from the users point of view.

These lessons were followed for the work in this chapter as well as the longitudinal investigations that take place in the next chapter. Specifically, small lightweight and high fidelity prototypes were built to demonstrate a single or small number or features of the framework. The framework was made available in phases to other developers as

components became more stable over time. Real components were used when practical.

7.4.1   Feasibility

A number of prototype applications were built to test key aspects of the approach and to ensure feasibility of the claimed features. Each of the following applications was built upon the MATCH framework, as described in this chapter, at various stages of completeness. The first prototype was to demonstrate the initial concept of the evaluation function model and to show that it could be used to switch between selections of output devices. In this example a continuous stream of temperature sensor readings from a Phidget [100] USB temperature sensor was displayed on a selection of three output devices (console, television & speech synthesis). This prototype ran on the MATCH architecture and was used to test performance and functionality and demonstrate the basic feasibility of the approach.

Figure 7.12: *Screenshot of First Prototype*

The second prototype was prepared in order to demonstrate the work of the MATCH project to at a conference demonstration session in Belfast and was presented by Louise Bellin. The purpose of this demonstration was to create a more compelling demonstration for a non-technical audience. In this prototype a SHAKE [233] device, shown in Figure 7.13, was used to monitor the user's movement over time and report anomalous situations (i.e. long periods of inactivity). The SHAKE is a wireless accelerometer controlled via a

152

Bluetooth connection and which is capable of sensing magnetic fields via a magnetometer and orientation via a gyroscope. SHAKEs also have a push button and can provide feedback via a pager motor powered vibration function. This demonstrated integration of a number of disparate interaction methods, incorporating a fully functional speech synthesis system, as well as exploiting evaluation functions for selection of both input and output interactions.

Figure 7.13: *SHAKE device*
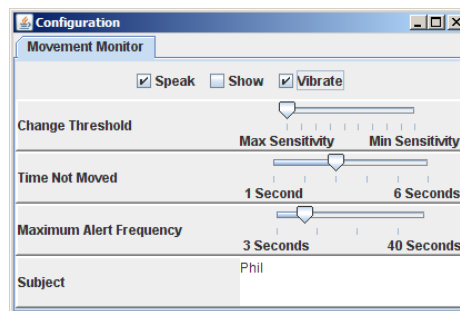
Figure 7.14: *Screenshot of Belfast Prototype*

Following on from this work it was necessary to create a more general tool enabling demonstrator applications to be rapidly created. This tool was built using the JGraph [1] library and primarily designed as a rapid application development tool which allowed the instantiation, combination and configuration of tasks and evaluation functions. This allows more flexible and rapid development of scenarios and testing of components and tasks.

The editor application, shown in Figure 7.15, allows the display of task or evaluation function specific panels within the editor so that both tasks and evaluation functions can be configured from within the editor application. This rapid application tool is not designed for use by end-users but instead by other application developers interested in working on
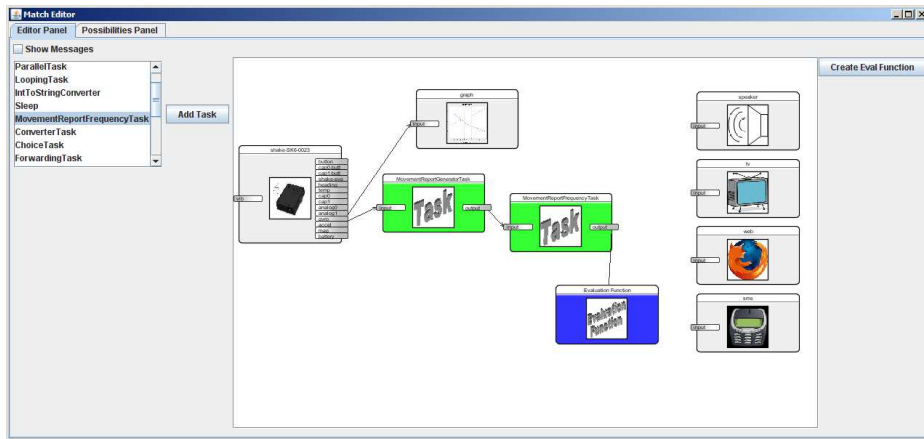
Figure 7.15: *Screenshot of Editor Tool*

the MATCH framework. Evaluation functions are used for controlling each of the input and output components which the task has been mapped to. Evaluation functions provide a `getJPanel` method which can be called by the configuration editor to allow users to configure evaluation function settings (by interaction with the provided JPanel).

One example of an application that was developed using the editor application was a demonstration for the SHAKE device; a wireless accelerometer with a gyroscope and vibration feedback. This application was composed of tasks and visualisation components suitable for use with the SHAKE device. Evaluation functions could be employed to discriminate the different sources of input and output for a *Movement Reporting* task that had been developed.

The SHAKE prototype provided five components; (i) *SMS text*, (ii) *Speaker*, (iii) *TV*, (iv) *Web Component* and (v) the *SHAKE* itself. The SMS text and TV components were mockups designed to demonstrate the functionality of the framework while the other three components were fully implemented. The SHAKE component included the inbuilt accelerometer (input), vibration motor (output) and a button on the case of the SHAKE (input) - thus providing three functions within a single component. The speaker was capable of playing wave files as audio through the sound card and associated speakers on the local machine while the Web Component would write the message it was delivered as a webpage which could be viewed in a regular web browser.

Only the HTTP Web Component accepts *Movement Report* messages directly, the channels available on each component were of the following types:

- *SMS Text* - SMS Text message which is displayed the message on a mock phone.

154

- *Speaker* - Wave files which are played through the local machines speakers.

- *TV* - Event messages triggering the temporary display of a symbolic icon overlaid on top of a mock TV screen.

- *Web Component* - Movement Report messages detailing the status of recent movement which would be written to a webpage.

- *SHAKE - Vibration Motor* - Event messages which triggered a short vibration through the SHAKE motor.

- *SHAKE - Accelerometer* - Acceleration messages at a rate of 60hz consisting of X,Y,Z vectors detailing the instantaneous acceleration of the SHAKE.

- *SHAKE - Button* - Event messages indicating when the button is pushed.

The two main application tasks were MovementReportGenerator (Movement Monitor) which accepted acceleration messages from the SHAKE and outputted regular Movement Report messages summarising the recent movement. The *MovementReportFrequency* task could be used to control the rate at which these messages were generated by acting as a filter.

Four further tasks were provided which could be used to convert the Movement Report messages into other forms that could be accepted by the output components (e.g. converting to an Event which could trigger the Vibration motor in the SHAKE or to an SMS message type which was accepted by the SMS component). These were:

- *MovementReportToSMS* - converts Movement Reports into SMS message format (accepted by SMS text component)

- *MovementReportToVXML* - converts Movement Reports into VXML (VoiceXML) message format

- *VXMLToWav* - converts VXML messages into wave files, using Speech Synthesis, which can be accepted by the speaker

- *MovementReportToEvent* - Sends Movement Reports as events

A final two tasks were used which could control the state of a boolean variable. These were the *MovementToBooleanState* task, which accepts Movement Report messages and sets the state to true if moved recently or false otherwise, and the *EventToBooleanState* where each received event inverts the boolean state from its current value.

Five evaluation functions were provided to allow the designer to select the components to be used. These were: (i) *UtilityFunction* which was hard coded to select either the

Web or Speaker component if available, (ii) *ComponentPreference* which provided a user interface to enter an ordered list of components where the highest rated available component would be used, (iii) *ComponentFilter* and (iv) *ChannelFilter* which both provided a similar user interface and selected each of the components or channel names which were entered and (v) *BooleanState* which accepted two other evaluation functions as children (*A* and *B*) and would execute the results of evaluation function *A* when the boolean state was true or evaluation function *B* when it was false.

Figure 7.16 shows one of many possible configurations for the SHAKE prototype. In the top left hand corner of the figure is the Movement Monitoring task which receives data from a component indicating movement and delivers an alert when movement has persisted for a period of time. The evaluation function has selected the accelerometer as the input source and the SMS text messaging service and vibration components as outputs; tasks represent any intermediate tasks required by the selected possibilities. This has a pleasing feedback effect as the act of shaking the SHAKE device causes a vibration to be returned directly to the device to allow the user confirmation that the physical movement was sufficient to trigger the SMS message to be sent.

This application highlighted how evaluation functions could be used within configurations which allow for feedback. The application partitions a larger configuration problem (configuring the entire application) into smaller, more manageable portions (configuring the connections to the two channels on the movement monitoring task).

7.4.2 Scalability

This section explores the factors that affect the scalability of the approach described in this thesis. There are three main factors that influence the scalability of this approach; these are (i) the number of components, (ii) the interconnectedness of the graph and (iii) the centrality of a graph. The impact of each of these will be discussed in turn in this section.

To determine the scalability of the Interaction Manager it was decided to conduct a number of performance measurements for the evaluation process and observe the effect that the size and complexity of the graph would have on the time taken to complete an evaluation under a variety of different sized graphs. A testing application was written which is capable of randomly generating Service Discovery descriptions of components, types and tasks. This service discovery data could then be used with the implementation of the Interaction Manager described in the previous section.

To conduct the measurements in this section the following approach was used; for each
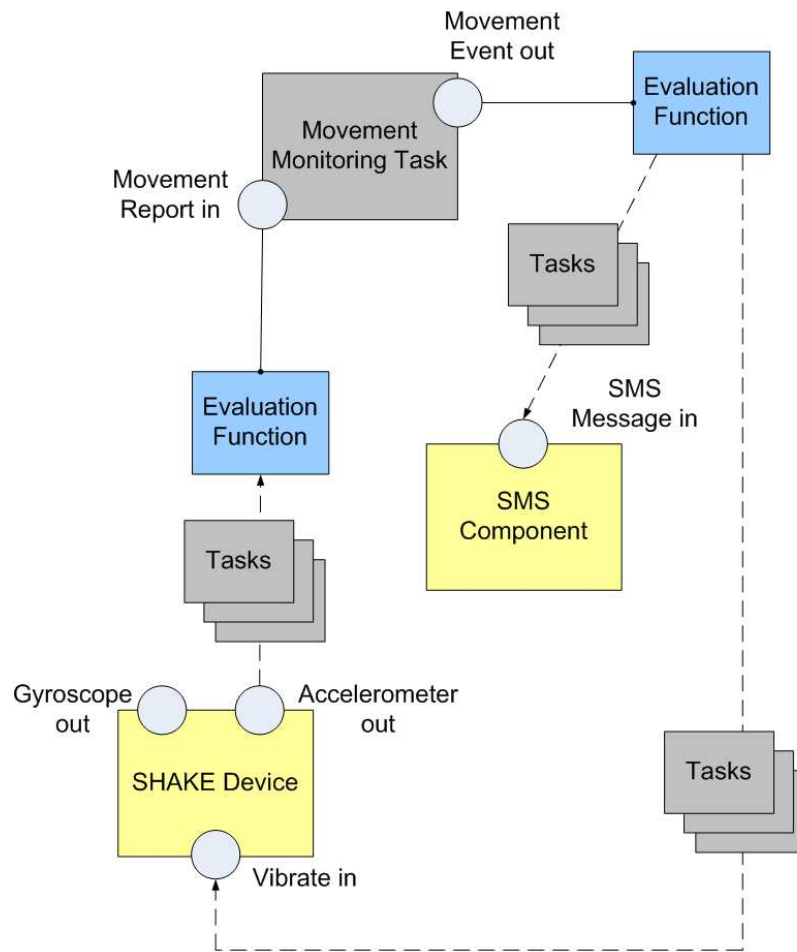
Figure 7.16: *SHAKE Configuration - Dotted lines indicate connections that have been made by the Interaction Manager as a result of evaluating an evaluation function*

datapoint required (a combination of a number of components, tasks and types), the desired number of each entity would be inserted into an empty service discovery database. For each task that had been inserted it would be randomly connected to two of the message types via its channels. Each component would then have a random number (between one and eight) of channels assigned to it with a random direction and random type. An application task would be created with an output channel of a random type.

The graph would then be constructed from the Service Discovery data using the application task as the root node. The graph would be traversed to build the possibility list and two evaluation functions, described below, were run.

Each of the three final steps was timed to an accuracy of nanoseconds as executed by a single thread on a 2.4Ghz Core 2 processor. As the graph is randomly constructed,

each measurement was repeated 10000 times and the samples averaged such that each datapoint is an indication of the true performance of the approach and to reduce the effect of individual graphs. The two evaluation functions used in this test were approval evaluation functions and were (i) the Endpoint approval function described previously in Section 7.3.4 (evaluation function 1) which checks that the terminator of the possibility is a valid component and (ii) the Direction Assertion approval function (evaluation function 2) which ensures that the possibility is valid in terms that all input channels are connected to output channels and vice versa by traversing the possibility. This demonstrates the performance of functions where runtime of the function is both independent (function 1) on the length of the possibility and dependent (function 2) on the length of the possibility.

An illustration of a graph generated is shown in Figure 7.17. Note that this graph is significantly smaller and less interconnected than most of the graphs used in this section as larger graphs become increasingly difficult to represent visually.

Worst case performance for the Depth-First search algorithm is bounded for an implicit (unknown) graph according to $O(b^d)$ where b is a branching factor and d is the depth to be searched in the graph. As the naive graph traversal algorithm used here visits every node of the graph using a variation on the Depth-First search algorithm it should be expected that this application will be bounded by this performance. This section finds the worst case performance for each of the algorithms used in this approach and shows that the worst case performance is still tractable for large problems (by the standards of ubicomp systems). However, it should be further noted that worst case performance is rarely encountered in real systems due to the emergence of logical patterns [232]. Specifically, ubicomp systems typically have very low branching factors in comparison to the randomly generated systems shown here.

7.4.2.1 Number of Components

The first factor to be explored here is the number of components in the framework. This factor acts as a constraint on the overall size of the graph that can be considered.

To explore this factor on its own the branching factor of the graph was controlled by fixing the number of type and task nodes to a fixed value (20 of each) and by varying the number of components between 10 and 200.

Figure 7.18 shows that the number of components has a linear effect on time. This can be explained by the observation that components are essentially leaf nodes within the graph. As such, their addition does not affect either the depth, or the branching factor
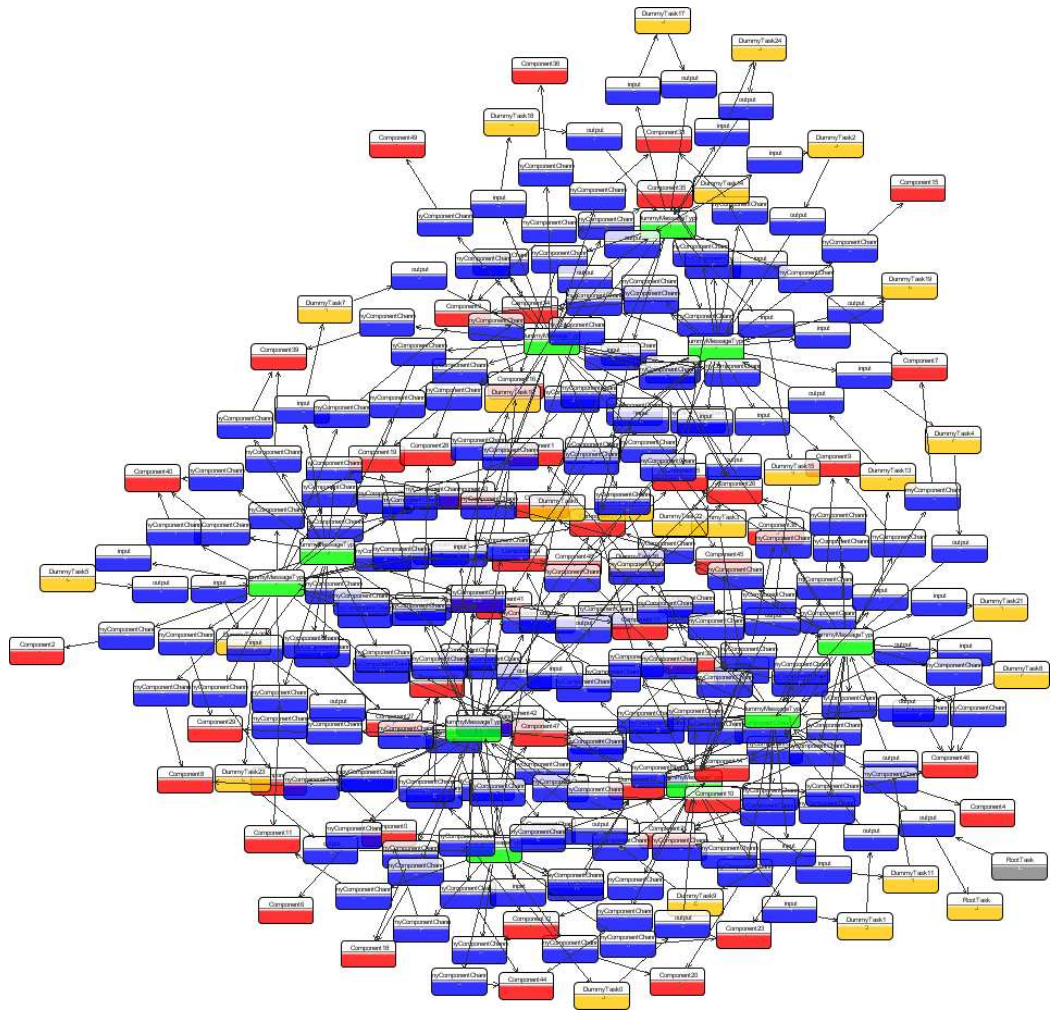
Figure 7.17: *A sample scalability graph generated by the above pseudocode for an application task (gray) with 50 components (red), 25 tasks (yellow) and 10 types (green) with a random (1-8) number of channels (blue) per task*

of the traversal algorithm and only adds a linear number of additional computations to each of the stages of the algorithm (accounted for by the additional possibilities that must be considered).

7.4.2.2 Interconnectedness

The interconnectedness of the graph is a measure of the degree to which nodes are connected to other nodes within the graph. This measure affects both the branching factor and the depth of the graph. Every additional connection between nodes adds additional

159
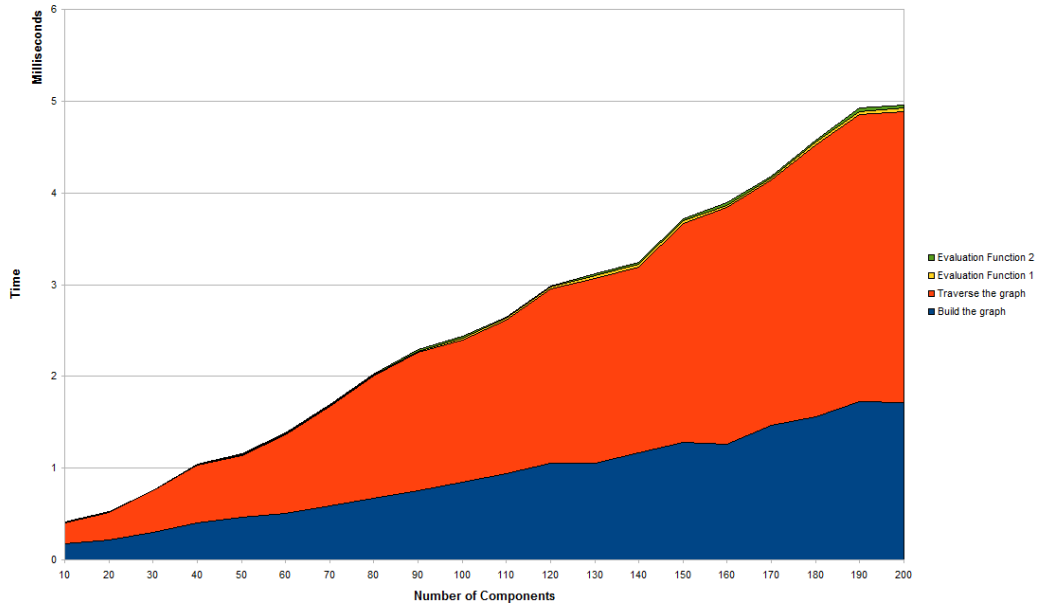
Figure 7.18: *Stacked chart depicting effect of number of components (x-axis) on evaluation time in milliseconds (y-axis)*

branches to the two nodes it connects as well as adding many more possible paths through the graph which increases the depth of searching required to fully explore the graph.

To explore this the number of components was fixed at 50 and the number of message types fixed at 20 while the number of connecting tasks was varied between 5 and 50. The expectation that the level of interconnection between nodes will have a large effect is confirmed in Figure 7.19. This is a result of the increase in the branching factor when a system is created where every component can be connected to every other component in multiple different ways. Fortunately high branching factors are rare with realistic systems [232] so this is not considered to be a large problem.

7.4.2.3 Centrality

Another factor that affects the branching factor of the graph is the number of central nodes within the graph. This is a node or group of nodes that has a high probability of being included in all possibilities. In the implementation discussed in this chapter, the message type has this effect as they are used to connect tasks, components and other types together. Reducing the number of message types within a random graph has the effect of centralising paths through the graph into a smaller number of central nodes which each have a higher

160
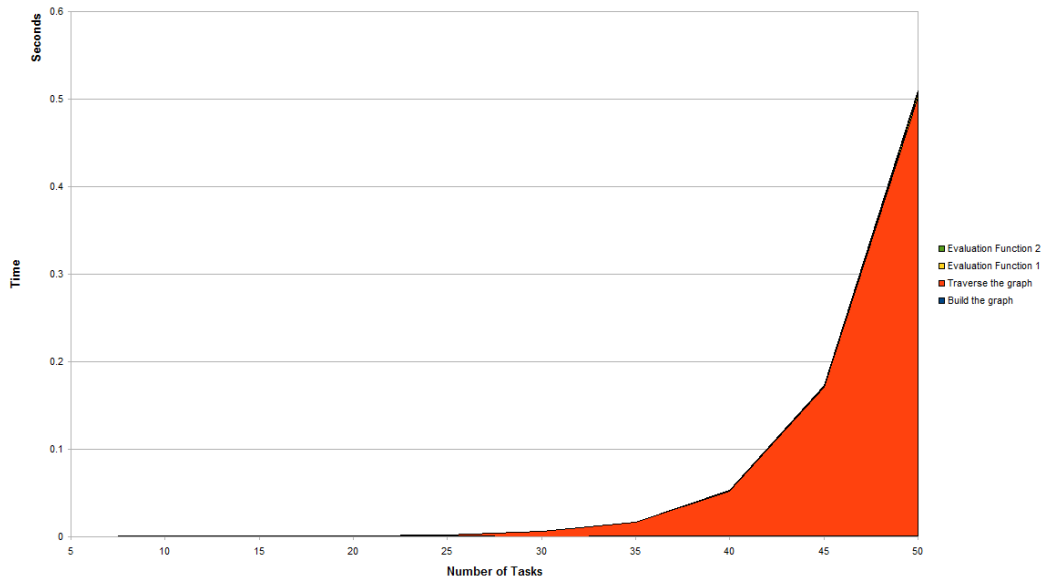
Figure 7.19: *Stacked chart depicting effect of number of interconnecting tasks (x-axis) on evaluation time in seconds (y-axis)*

branching factor.

An increase in the number of central type nodes shown in Figure 7.20 has the effect of decreasing the branching factor showing the expected result that the number of centralised nodes has an inversely proportional effect to running time and thus indicating that more distributed graphs have superior scalability.

7.4.2.4   Discussion

Even with the effect of high interconnectedness, the maximum average time to execute the entire evaluation was still under one second for any combination of datapoints shown here. The graphs generated here represent the worst case performance for this approach as real world systems can be expected to exhibit lower branching factors than randomly generated graphs due to the fact that not all data types can be reasonably transformed into all other data types. As such, these results are a minimum level of attainable scalability.

This result shows that the approach is scalable in terms of the absolute number of components available within the framework. Scalability is greatest for graphs with low branching factors - a result which is in agreement with expectations. Approaches to increase the scalability of the traversal stage of the approach further are discussed in Chapter 9.
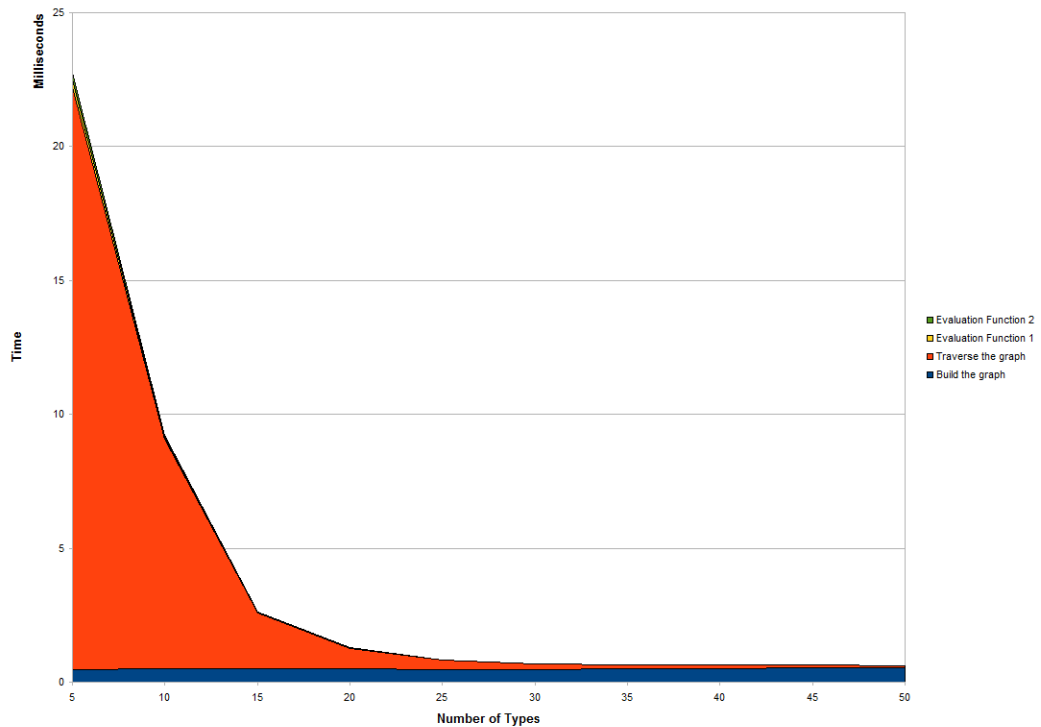
Figure 7.20: *Stacked chart depicting effect of number of central nodes (x-axis) on evaluation time in milliseconds (y-axis)*

7.4.3 Flexibility

A number of other people have been interested in using it in order to build their own tools and applications.

The citation of the work discussed in this section should not be taken as implying that these systems build on the MATCH framework constitute thesis contributions. Rather, each of the projects discussed below builds on top of the approach described in this thesis in order to implement a specific application suited to their own project or thesis. Projects are discussed in approximate chronological order.

The success of each of these projects indicates that the approach, and the associated framework implementing it, is both useful to other application developers, usable by them and flexible enough to create a large range of applications.

7.4.3.1 Speech Component

As part of the MATCH project a speech synthesis system was developed at the University of Edinburgh by Neil Mayo and Ravi Vipperla. This work was based on the proprietary Cerevoice speech system developed by CereProc; itself a company spun off from the Edinburgh-Stanford Link speech research fund. Cerevoice is provided on a gratis basis for academic research and is used by the Edinburgh speech labs as part of their approach to developing speech interfaces.

The Cerevoice system used, and extended, by Neil Mayo and Ravi Vipperla is based around a core runtime library built in C with additional Python wrapping and functionality. This system was further wrapped using the Jython runtime engine and JNI (Java Native Interfaces) was used to allow access to the Python and C routines from within a Java Virtual Machine.

The speech synthesis system was then integrated into the MATCH framework using the "Heather" voice. Integration was accomplished by developing a MATCH task which was capable of accepting VoiceXML segments to be transformed into speech on one channel and outputting synthesised Wave files on another channel. To perform the actual speech synthesis the task calls the appropriate method within Cerevoice to obtain a synthesised speech sample. This task can then be coupled with an associated Speaker component capable of playing the speech samples on the local machine.

Although Cerevoice itself supports multiple voices, the academic licensing limits the availability of voices to the Heather voice. The Heather voice is a female Scottish adult and is used for a variety of purposes; including being funded by the Scottish Government[2] to provide a Scotland wide schools license for Heather to allow Scottish curriculum resources to be spoken using a Scottish synthetic voice.

The approach used to integrate Cerevoice into MATCH allows for the use of multiple voices were it available by searching for available voices and registering one task to perform synthesis for each voice detected.

This integration demonstrated that it is possible to take pre-existing applications and services which were not designed with this approach in mind and integrate them into the MATCH framework and to incorporate them as tasks and components which can be used within possibilities.

---

[2]http://www.thescottishvoice.org.uk/About/

7.4.3.2 Phidget Sensor Components

This theme was extended by Alex Walker who was a summer project student within the University of Glasgow for ten weeks in the summer of 2007. Among other roles, Alex investigated the use of Phidget [100] devices for sensing and multimodal interaction.

Phidgets are collections of simple sensors (such as distance/range, force/pressure, touch, motion and electrical) and actuators (primarily servo and stepper motors and electrical switches). Phidgets are controlled via a USB Interface board which may additionally contain an LCD display. Each sensor is connected to a port on the interface board which delivers or receives an analogue signal quantized into 256 values which can be set or retrieved using a Java based API.

Alex developed a small number of components within the Match framework which utilised the Phidget sensors and actuators and which were used to build applications for investigating the utility of Phidget-type sensors within the home.

The components included a sensor for Phidget based Accelerometers and RFID readers as well as providing a number of output devices for both sensors; including Servo actuated motors, the Phidget LCD display and a small selection of GUI based display techniques. The work conducted by Alex showed that it was possible to provide multiple different output devices and modalities for a single data source within the MATCH framework.

7.4.3.3 Daily Activity Visualisation

During the same period of Alex Walker's work with the MATCH framework, ChuanJun Wang built a set of visualisation components for presenting information about a user's daily activity as his MSc project submitted in September 2007. This was built using both the MATCH framework and the Replayer [158] toolkit. Replayer is a visualiation suite designed to help in the evaluation and design of ubiquitous computing to allow visualisation of complex and multi-dimensional data developed by the Equator group within the University of Glasgow[3].

ChuanJun built an Activity Monitor application which allowed four visualisation components for visualisation of real time and historic movement, sleep patterns, light and temperature information in a collection of rooms (living room, bedroom, kitchen etc) as shown in Figure 7.21. The supplied visualisation components could display data for each

---

[3]Originated in the PhD work of Paul Tennent and extended by Dr. Alistair Morrison

of these activities summarised data as events or by time in barchart, linechart or scatter plot format as shown in Figure 7.22.

Figure 7.21: *Room Layout*

Figure 7.22: *Summarised Data*

The sensors chosen by ChuanJun were very high level and it was not part of his project to develop appropriate sensors to detect either movement or sleeping even though his application was designed to show this type of information given a suitable sensor type. Instead, ChuanJun used virtual software sensors to simulate the output of a physical sensor and created these as components within the MATCH framework.

In order to demonstrate that his monitoring application was capable of dealing with data from real sensors, ChuanJun utilised the temperature sensing capability of the Phidget interface, developed during the same period by Alex Walker, and demonstrated that it was easy to change the virtual sensors for real ones using the component model used by the MATCH framework.

ChuanJun developed a monitoring application which showed the integration of multiple disparate types of data which could be processed and summarised within the MATCH framework and was awarded an MSc in IT based on his thesis [228] which described this application.

7.4.3.4 End User Programming Environment

In April 2008, as his final MSci project, Usman Khan developed an end user programming environment intended for home users to be able to configure and customise a home care system to their needs. This work was undertaken using the MATCH framework.

Usman's research was interested in investigating the requirements for end user programming of a home care system using the visual programming metaphor. The aims of this work were to create a user interface which allowed users to determine the available services and configure them within a ubiquitous computing application domain and use it to identify and explore the major issues associated with end user programming applications.

A prototype was created within the MATCH framework which would allow for four typical applications that users might build and configure. These were: (i) a Night Wandering Alert, (ii) a Symptom Manager, (iii) a Temperature Alarm and (iv) a Door Camera. Each of these example applications had a number of configurable parameters as well as a selection of available interaction components with which the application could operate.

The applications described above were modelled as tasks within the MATCH framework which utilised evaluation functions in order to specify the appropriate connections to be made between tasks and configuration components. Usman's application used evaluation functions in order to connect components to the application tasks that he had created; these connections were initiated by the use of Usman's visual programming tool.

An example of this editor in use is shown in Figure 7.23. Initially Usman had planned on repurposing the Jigsaw [109] user interface and using it with the MATCH framework but decided that it would be simpler for his purposes to develop his own editor using the OpenJGraph library in order to create a Jigsaw-like user interface. The editor that Usman created was a motivator for the development of a more general purpose editing
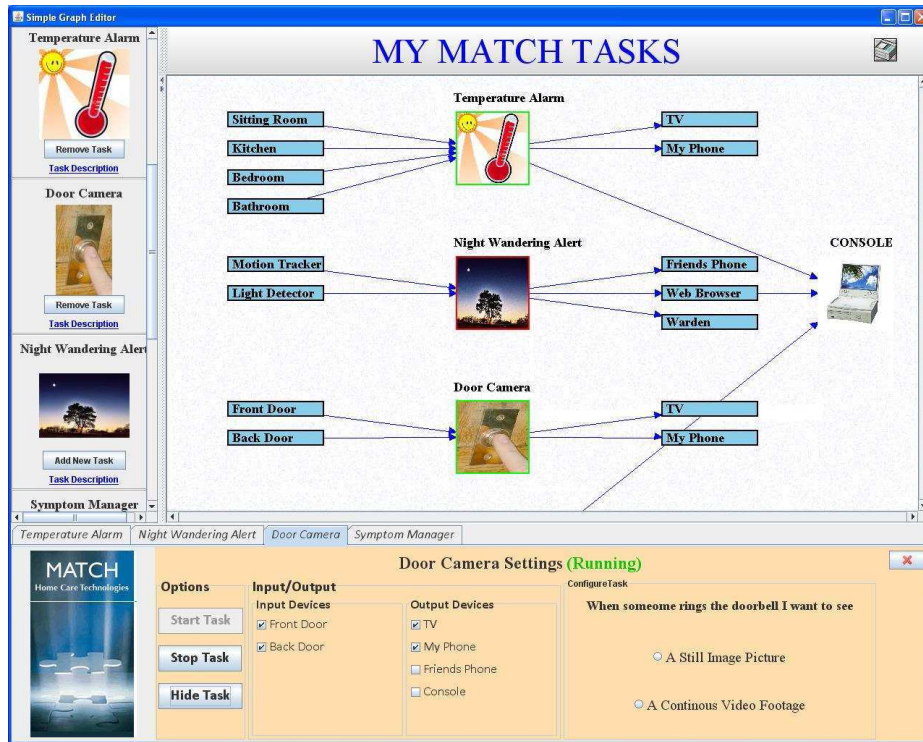
Figure 7.23: *Editor Application*

application which was built shortly after Usman's work using the JGraph library as shown in Section 7.4.1.

Finally, Usman used this tool to conduct a sets of evaluations of end user programming applications ranging from usability studies of his tool to a comparison study of his prototype tool against other context aware applications.

Usman demonstrated the ability to use tasks and evaluation functions to implement an end user programming environment which allowed users to configure a home monitoring system and was awarded an MSci in Computing Science based on his thesis [119].

7.4.3.5 Multimodal Reminder System

Lauren Norrie was the recipient of a summer studentship between June and August 2008 awarded by the Faculty of Information and Mathematical Sciences on behalf of EPSRC. Lauren's project focused on the area of multimodal interaction techniques such as speech, non-speech sound, gesture and tactile interaction and focused on the domain of reminders in the home (such as appointments or medication reminders).

This work relied upon the notion that reminders should be configurable and adaptable to the home environment such that the modality they are presented in can be changed depending on context. She used the MATCH framework as a base in order to satisfy this requirement.
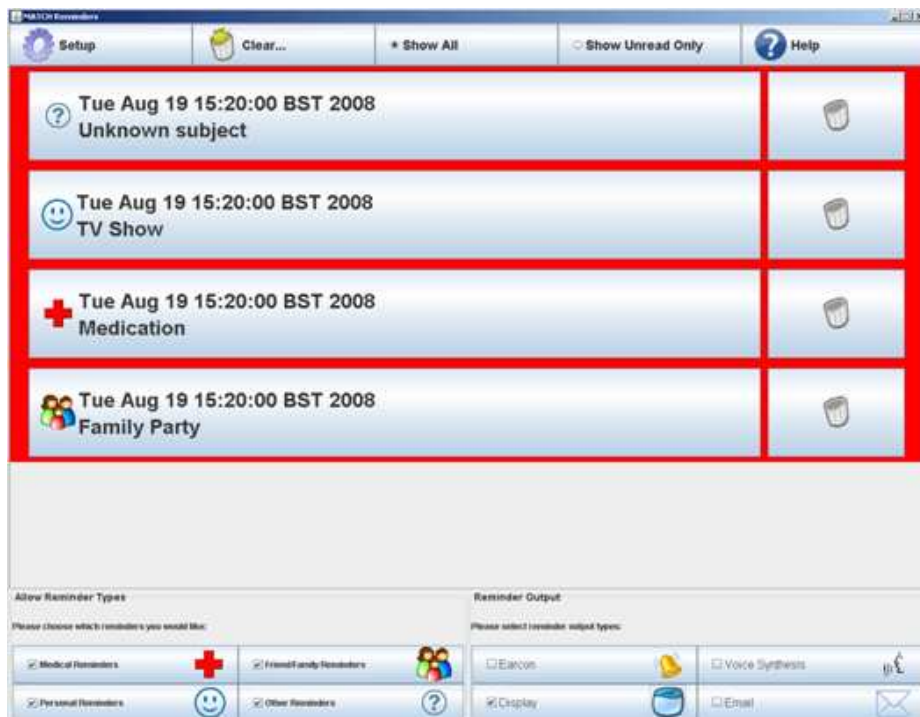
Figure 7.24: *Reminder System*

Lauren's application, as shown in Figure 7.24, allowed users to receive reminder notifications generated by a Google Calendar account and to select a collection of destinations. Laurens application allowed filtering of incoming types of messages as well as selection of output modalities and accomplished this by using evaluation functions to select which reminders should be processed as well as which output devices should be used (options of Earcons, Speech Synthesis previously developed by Neil and Ravi, Graphical Display and Email).

Lauren made three direct contributions to the MATCH framework in addition to her project; she created a substantial amount of API documentation and "How to" tutorials for later users of the MATCH framework and created a Google Data and Email component for MATCH which was later used in subsequent prototypes to retrieve calendar reminders and dispatch Email notifications (See Chapter 8). Finally, Lauren was asked to streamline the deployment of the MATCH framework by creating an automatic "release" packager to create an executable based on the existing Eclipse workspaces - a task which had been

tedious until that point.

By using the MATCH framework, Lauren did not need to worry about the implementation details of the output modalities nor build logic to create pipelines between her resident reminder task and the input and output sources as the Interaction Manager handled this complexity for her - allowing her to concentrate on the project aims.

7.4.3.6 Home Automation Components

Claire Maternaghan is a PhD student at the University of Stirling in Scotland. Claire's research [141] is focused in home automation and component architectures, specifically component models similar to Service Oriented Architectures.

Prior to beginning her work on her PhD, Claire undertook a small project to create a small number of components for the MATCH framework in order to investigate which devices she wished to use for her own PhD. This was primarily an exploratory exercise and Claire made use of a number existing MATCH components as well as designing and creating several of her own.

The first component she developed was the Nabaztag [4] Rabbit produced by the company Violet. The Nabaztag is programmable interactive avatar which communicates with the Violet servers and is capable of communicating with you by moving its ears, playing music and speaking through an inbuilt speaker, through LED lights on the front and is capable of detecting RFID tags. An example of a Nabaztag rabbit is shown in Figure 7.25.

Claire used the MATCH framework, and editor tool, to create a Nabaztag component which she could use to experiment with. To accompany it she created a Nintendo Wiimote component which could be used as an input source. The Wiimote is a consumer accelerometer paired with the Nintendo Wii[5] gaming console and can be used as a gesture and physical button input peripheral shown in Figure 7.26.

Using these two components, and with the existing MATCH components for SHAKE based[6] accelerometer input, speech output, the components she had developed herself and the MATCH framework and editing environment, Claire was able to explore the range of component devices that would be of relevance to her future PhD work.

In addition, Claire created an onscreen keyboard Java application which was later extended and incorporated in the investigation in Section 8.4.

---

[4] http://www.nabaztag.com accessed 2010

[5] http://www.nintendo.com/wii - accessed 2010

[6] To be described in Section 8.3.2

Figure 7.25: *Nabaztag - Courtesy Violet*

Figure 7.26: *Wiimote - Courtesy Nintendo of America Inc.*

Claire demonstrated that the MATCH framework could be integrated with a variety of very different commercially available input and output modalities which were not designed specifically with the MATCH framework in mind.

7.4.3.7 Ontology-based Service Discovery

Dr. Liam Docherty was a student at the University of Stirling who used the Match framework as a deployment target for his PhD topic studying Ontological Service Discovery (August 2008). During his work he created an Ontology Registry service which was deployed within the MATCH framework and provided a protocol independent approach

for describing reasoning over the devices within the home and storing policy based data such as user preferences and rules.

An initial vocabulary for homecare was developed in the form of the Home Network Ontology Stack (HNOS) [229] which supports the application of ontology descriptions, using the Web Ontology Language (OWL) [17], to components in the MATCH framework. This thereby supports protocol-independent and supplier-independent descriptions of components.

Liam's work was designed to provide an OWL based ontology description framework which allows existing (non-ontological) description techniques to be incorporated and to support a semantically-rich discovery process. Liam's ontology registry was deployed as an OSGi bundle within the MATCH framework and was integrated with a method compatible API and was able to replace the simpler hashmap-based Service Discovery subsystem, providing additional semantic reasoning over available components and tasks.

This service allows evaluation functions (and other services as required) to perform semantic reasoning over the available components. Liam was able to use the MATCH framework to show the flexibility and usefulness of his approach in terms of "real-world" perspectives. However, some third party libraries used within the HNOS implementation imposed large performance penalties on addition or removal of objects within the registry which prevented Liam's service from completely replacing the simpler Service Discovery model used in the deployments to be described in Chapter 8 - however, a two-level approach could be considered for future work in this area where a simpler hashmap based implementation can be responsible for rapidly changing information such as device availability while the Ontology registry can be used to provide semantic reasoning over components as an additional service.

Liam was awarded his PhD based on his thesis [55] which used the MATCH framework to demonstrate; (i) the ability of the ontology language to describe components abstracted from protocol or vendor specific descriptions, (ii) the ability of the ontology vocabulary to adapt to user and system requirements at run time, (iii) the ability of the ontology vocabulary to describe interaction details of components and (iv) the ability of the ontology registry to provide logic-based discovery environment. This demonstrated that the underlying components within the MATCH framework could be extended and enhanced to provide additional semantic reasoning.

7.4.3.8   Verifying Interoperability Requirements in Pervasive Systems

Another avenue that the MATCH framework has been exploited is in the, EPSRC funded, *Verifying Interoperability Requirements in Pervasive Systems* (VPS) project. The VPS project is a collaboration between the Universities of Birmingham, Glasgow and Liverpool in the UK which aims to use deductive methods, such as model checking, and quantitative techniques, including probabilistic and performance analysis, in order to tackle the problem of verifying pervasive systems. The definition of pervasive systems used within VPS is a general class of systems that can sense their physical environment and adapt their behaviour accordingly [168].

Specifically, the aims of the VPS project include developing techniques and frameworks for modelling interoperability requirements in pervasive systems for interaction, performance and security, and evaluating the techniques on significant case studies in realistic application domains.

In their first paper [4] the VPS project has used the MATCH framework as a pervasive case study and to propose formal verification approaches for pervasive systems.

In this paper they create a formalisation of the requirements of the properties of the MATCH system using a combination of the access control language RW [102] and standard linear-time temporal logic (LTL) [70]; showing that a single approach to formalisation of the requirements is insufficient to fully capture the properties of a pervasive system and that a combination of approaches is required. This paper does not specifically refer to the model approach for evaluation that is described here; instead it focuses on higher level properties of the framework.

In a second follow up paper [31], the VPS project proposed an approach to tightly coupled verification of pervasive systems. In this paper they used model checking and SAT solvers to reason about the configuration model presented in this thesis. The vision of the VPS system as described by Calder et al. is detailed below and presented in Figure 7.27.

> *The key feature of our vision is that modelling is tightly coupled with system development and configuration. This is not a waterfall model: activities are concurrent and moreover, while four agent roles are indicated, they may be overlapping or conflated. Briefly, activities are as follows. The end users configure the system, and when configured, (possibly different) users interact with the system, as system and users require, according to the context. The configuration is not static, but may be changed repeatedly. Log files are a representation of the configuration process and are generated by a live*

system. The formal model depends upon what kind of analysis is required (e.g. functional behaviour, security, performance, etc.) and it is configured, according to the log files. The model is analysed; the verification results may inform understandings of the end user, the configurer, the designer, and the modeller, though in different ways. For example, the user develops a better cognitive model, the configurer understands how to improve his/her rules, the designer develops a better interface, and the modeller gains insight in to how to modify the model so that verification is more efficient.
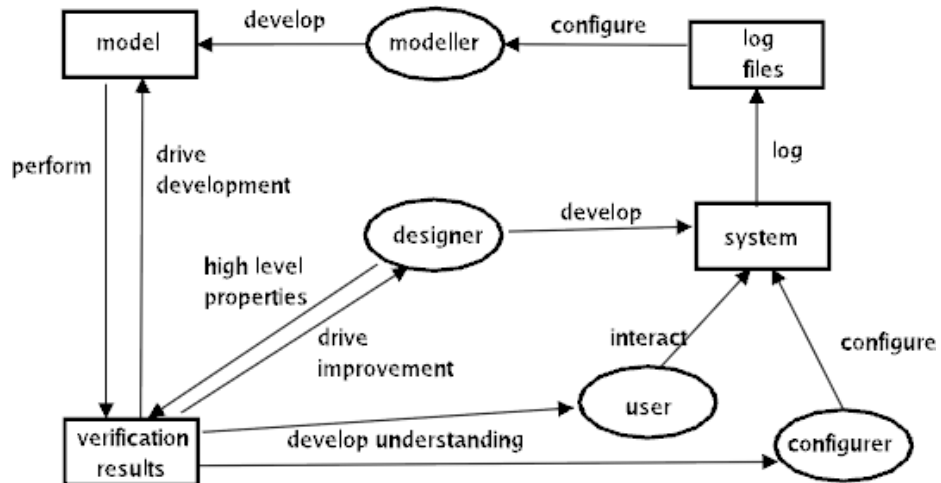
Figure 7.27: *VPS - Tightly coupled verification: configurable systems and configurable models*

Modelling of the MATCH framework was undertaken using the Activity Monitor application which is described in the upcoming Section 8.4.2. Chris Unsworth, a postgraduate researcher on the VPS project, developed a general purpose model of the MATCH framework using Promela [108]; a high level, state-based, language for communicating concurrent processes. This model is built directly from log files detailing the state of evaluation function configuration from a running system. The log files specify the available possibilities, combination of evaluation evaluations, tasks running and results of each evaluation function execution at each point in time of the system. Using this log file the evaluation function rule set is expressed as an informal natural language rule set and then expressed as Promela statements.

A number of situations were identified where formal verification can benefit configuration of interactive systems:

- **Redundant rule detection** - Evaluation function configurations may have some

overlapping or repeated definitions between tasks and it is advantageous if such redundant configurations can be detected to provide feedback to the user or to remove them from the active configurations

- **Modalities** - Input and output components can be classified by modalities and the acceptability of such a system may depend on correct use of different modalities. It is possible to validate that evaluation functions are correctly choosing only appropriate modalities for the user in question; i.e. visual output devices should be avoided for severely visually impaired users.

- **Priorities** - Some messages or interactions may be deemed of higher importance than others and it can be useful to check that there is no overlap between forms of high and low priority messages.

The Promela model was then used with the model checker SPIN which represents properties expressed as logic LTL (linear temporal logic) which Promela rules are a form of. Redundancy checking was tested using a realistic rule set taken from actual log files of the MATCH framework and each configuration was tested in-turn for redundancy. Using the SPIN model checker the verification times ranged from 12 minutes to 34 minutes with a search depth of 4 to 6 million and exploring between 65 and 100 million states.

Since the VPS project aims to provide real-time verification of pervasive systems another approach had to be considered in order to provide feedback to a system configurer in real-time. The second approach was to employ a specialised SAT solver [68]. SAT solvers check the satisfiability of formulae written in disjunctive normal form and, although in general NP-complete, are highly efficient for many practical applications.

A Java program was used to automatically generate SAT models from the log file describing evaluation function configurations and then solve the SAT problem using the open source SAT solver miniSAT and checks the configuration set for redundancy. Using the SAT solver, each individual SAT model required 15 milliseconds to solve; offering a significant improvement of 5 to 6 orders of magnitude for redundancy checking over the Promela method.

The work undertaken in the VPS project aims to tightly couple design, use, configuration, modelling and verification and to enable automation of these processes. Their work has shown that the interaction model presented in this thesis is suitable for a formal verification approach of configuration expressed as evaluation functions.

7.4.4 Applying the model to other systems

To demonstrate the broad applicability of the approach described in this thesis, two very different exemplar systems are examined in the form of worked walkthroughs showing how the approach described here can be applied to these existing systems and how it can add flexibility to systems which do not natively support evolution.

The intent of this section is to show that the contributions in this thesis are not limited to the implementation that has been discussed in this chapter and that the techniques can be applied to other systems and frameworks.

7.4.4.1 OpenInterface

The OpenInterface (OI) project[7] is an EU IST (European Union Information Society Technologies) program funded research project that addressed the development of multimodal interaction technologies involving augmented devices in a ubiquitous computing environment.

One key contribution of the OI project is the OpenInterface framework [202] which is used to develop different interaction possibilities. The OI framework contains a graphical development tool known as OIDE which allows components to be assembled to specify pipelines from collections of components which the framework provides.

The OpenInterface framework is very similar to the MATCH system in terms of pedigree. Both systems have a notion of components which can be connected to each other via typed communication channels. OI supports components which can be dynamically instantiated, in fact all components are such in the OI framework, and where the components connections can be changed manually by the user. Configurations can be saved to disk to allow changing between configurations without having to manually specify them every time. The Open Interface framework has a notion of the Open Interface Repository which is capable of providing component information using OIMCDL (OpenInterface Component Description Language) and fulfilling the role of Service Discovery.

The OI framework does not have any native notion of reconfiguration other than that performed manually by the user. This section will take a sample application within the OI framework and add the concept of evaluation functions to derive a more powerful and flexible system while retaining the conceptual notions from the OI project.

---

[7]http://www.openinterface.org/

Figure 7.28: *OpenInterface Application - Annotated - Courtesy OIDE Development Team*

176

Figure 7.28 shows an annotated screenshot of the OpenInterface OIDE tool. Within this image each component is represented as a rectangle with the name of the component and a collection of channels; input channels on the left of the component and output channels on the right. Channels are typed in OI but not indicated in the Figure (excepting some channels named "StringOutput" for example). Arrowed Connections between channels indicate those two channels have been bound together.

In this example, the OpenInterface framework has been configured to realise an application involving the SHAKE device [233] (a wireless accelerometer with push button), Speech Recognition, Google Earth, Nabaztag Rabbit and Fingertip Light interaction. The overall aim of this application is to implement a multimodal interface to Google Earth [30] using speech commands to navigate to cities on the map and perform actions such as zooming and to additionally set a placemarker if the SHAKE is shaken at the same time. The push button on the SHAKE can be used to disable or enable the speech recognition engine. A fingertip based touch device is used to manipulate the map via panning, tilting etc. Finally, audio feedback regards navigation was delivered using a Nabaztag rabbit shown in Figure 7.25.

Within the OIDE tool there are a number of intermediate components (e.g. the OILightIn-teractionGoogleEarth component and the TupleFeeder) which are not involved in the logical operations but which are there primarily to facilitate connection between one device and another.
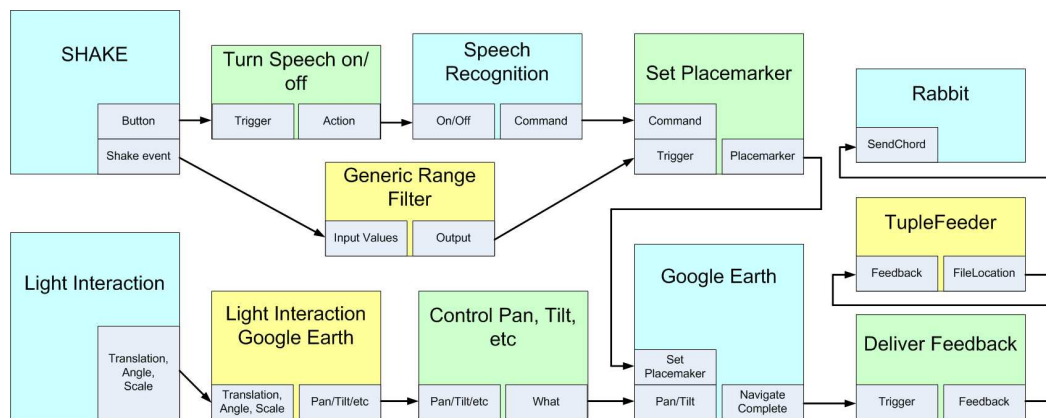
Figure 7.29: *OpenInterface Application - Simple Form*

Figure 7.29 shows a simplified form of the OI application designed to remove unnecessary details which are not of interest in this examination of the system and to make discussion of the system easier. Furthermore, components performing logical application task-like functionality (green) are distinguished from general purpose components representing

devices (blue), which are further separated from intermediate components which are used primarily to connect components together (yellow).

Figure 7.29 specifically highlights the logical operations taking place within the OI application. These are: (i) activation/deactivation of the speech synthesis on some trigger, (ii) setting a placemarker on receipt of both a command and a trigger, (iii) controlling pan/tilt etc, and, (iv) delivering feedback that a command has taken place.

The next step is to group the intermediate components with their associated components and highlight places that choices of components for each of the logical tasks can be made.
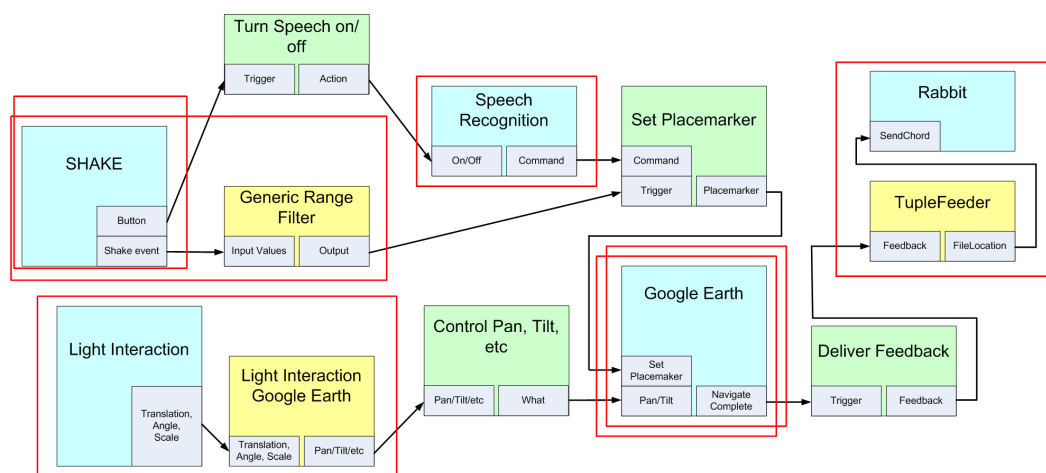
Figure 7.30: *OpenInterface Application - Grouped*

Figure 7.30 reproduces the previous figure and highlights the choices that have been made for each of the logical tasks by surrounding them in a red box. Note that two of the components have been highlighted twice; Google Earth is the choice of destination for the pan/tilt commands as well as placemarkers and the SHAKE component is used both to turn speech on and off and to signal placemarkers.

Each of the red boxes could be represented as a possibility for the attached application task. In the OI framework, the possibilities are very simple as the design consideration has been to minimise intermediate components to make manual assembly of components easier.

Figure 7.31 shows one of the application logic tasks in detail with two evaluation functions applied and showing a variety of different possibilities which would be available using only components (with the exception of the Feedback to Ear intermediate component) already available within the framework. This application shows how the source and destination for feedback can be configured at runtime without the need for manual reconfiguration of the system and allows the use of the approaches discussed in Chapter 6.
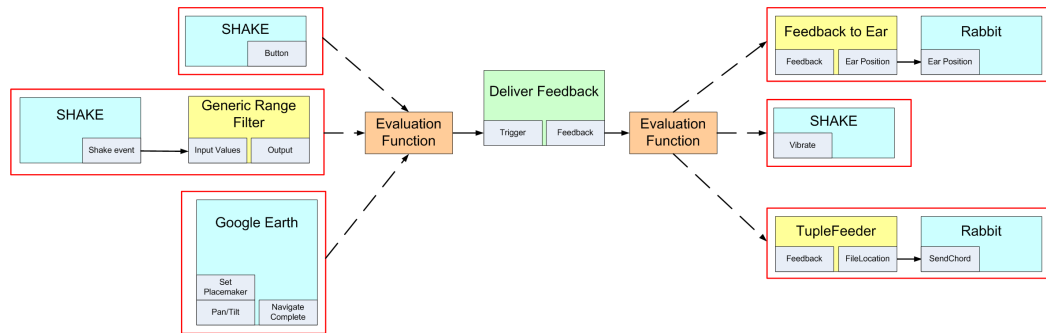
Figure 7.31: *OpenInterface Application - Possibilities*

7.4.4.2 ASUR / ASUR-IL

ASUR is a modelling framework which is designed for model-based description and development of physical and digital entities in a mixed system, modelling the user, physical devices and software components and the relationships among them [61]. ASUR models describe user interaction with a system and its related physical artefacts.
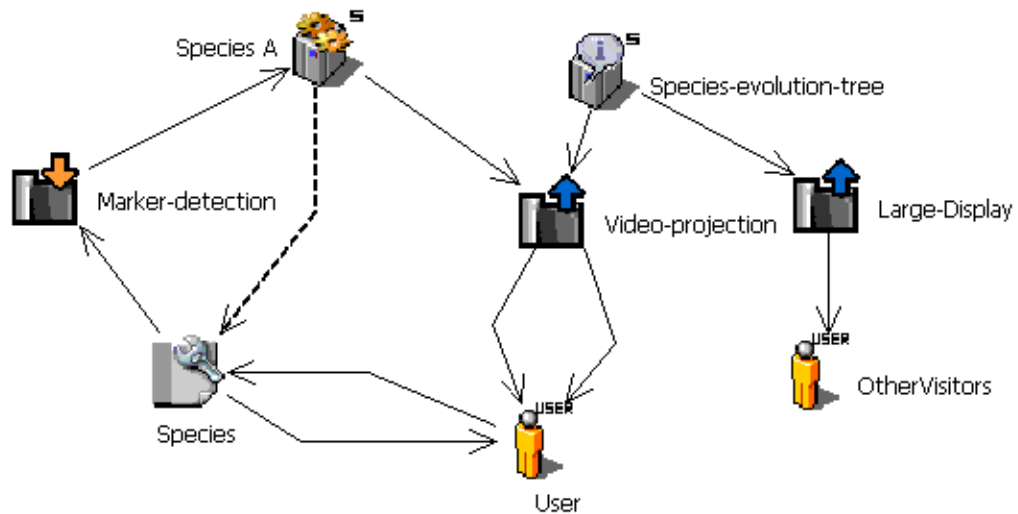
Figure 7.32: *ASUR museum model*

Figure 7.32 shows an ASUR model constructed by Gauffre et al. [87] which allows visitors to a museum to discover about species evolution through interaction with a large touch screen tabletop display and tangible species markers as shown in Figure 7.33.

ASUR-IL [60] is a complementary model which is used to describe software structures that implement ASUR specifications. Transformations between ASUR and ASUR-IL
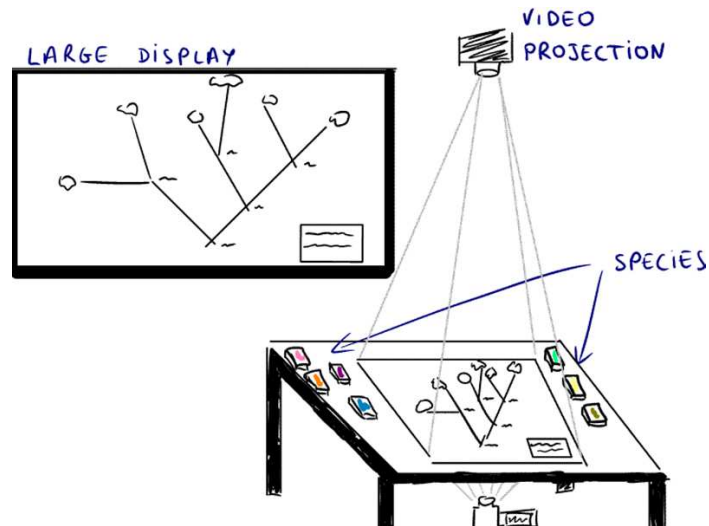
Figure 7.33: *ASUR museum design*

are expressed through a collection of static transformation rules [60] which transform modalities in ASUR into adapters which specify which default devices and APIs should be used. The result of such an application of these transformation rules is shown in Figure 7.34.
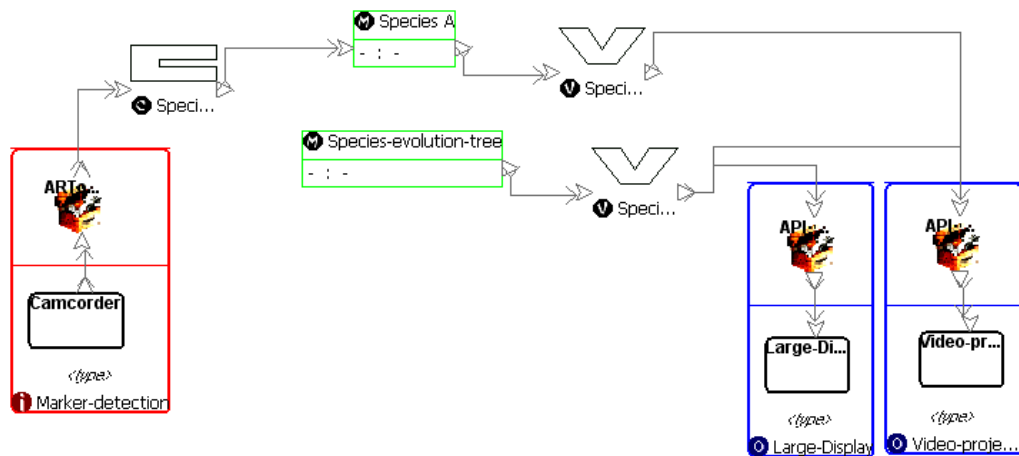
Figure 7.34: *ASUR-IL transformation of museum model*

During the transformation from ASUR models into ASUR-IL the designer is required to specifically choose components to be used from the set of available components. One future aim of the ASUR approach is to identify properties related to the quality of the interaction between a user and a mixed environment and include them into the evaluation process in which ASUR-IL adapters are selected [87] to increase the ability to evaluate the quality

of each interactive situation. It is in this area that evaluation functions can be particularly useful.
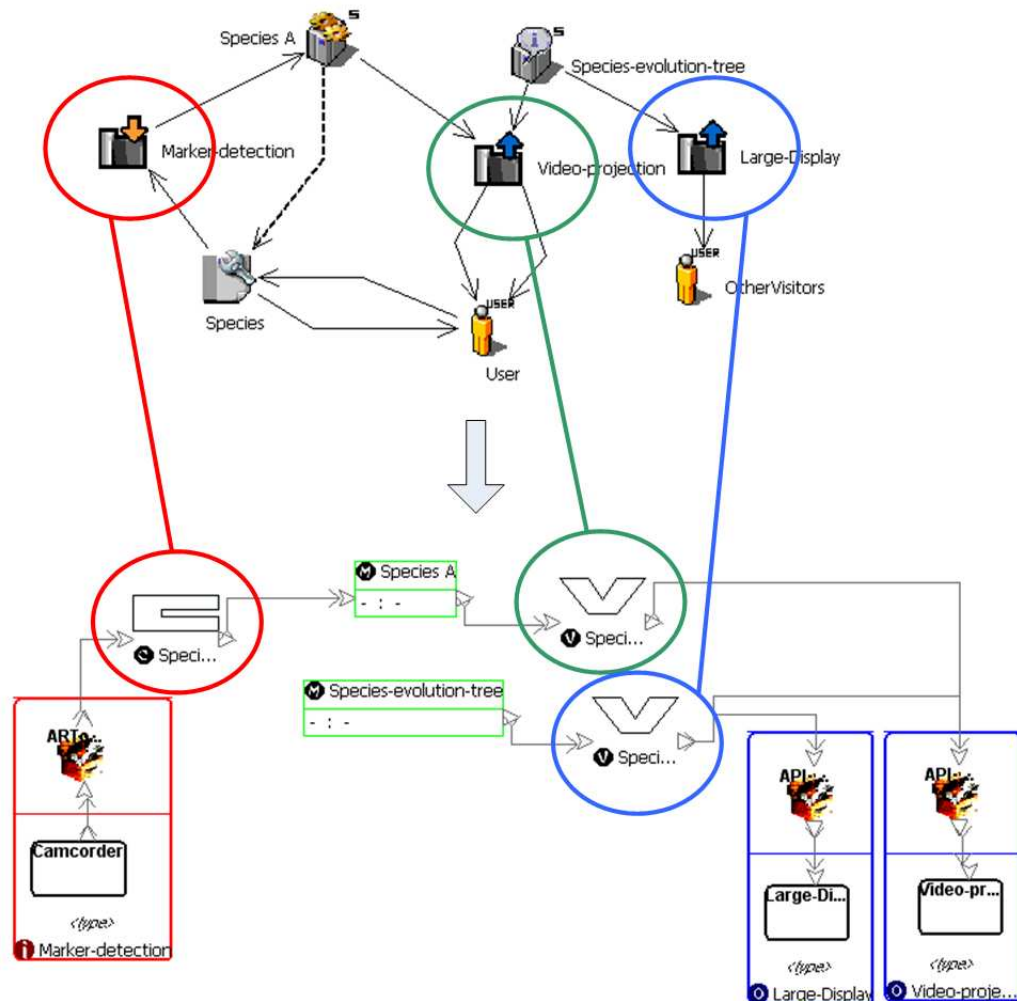
Figure 7.35: *Comparison of ASUR to ASUR-IL*

Figure 7.35 highlights the adapters within the ASUR configuration and their transformation into ASUR-IL adapters. Note that each of the adapters has been instantiated with an implementation (default device + API) which has been selected by the transformation rule. The API + device is equivalent to a possibility; instead of selecting the default device and allowing explicit modification by the developer later, the inclusion of evaluation functions would the choice of devices to include other properties which affect the success of the interactive experience; for example choosing video projection if a large audience is present and interactive PDAs when a small audience is present. Using evaluation functions allows the incorporation of other properties which affect the success of the interactive experience

to be included in the choice of device.
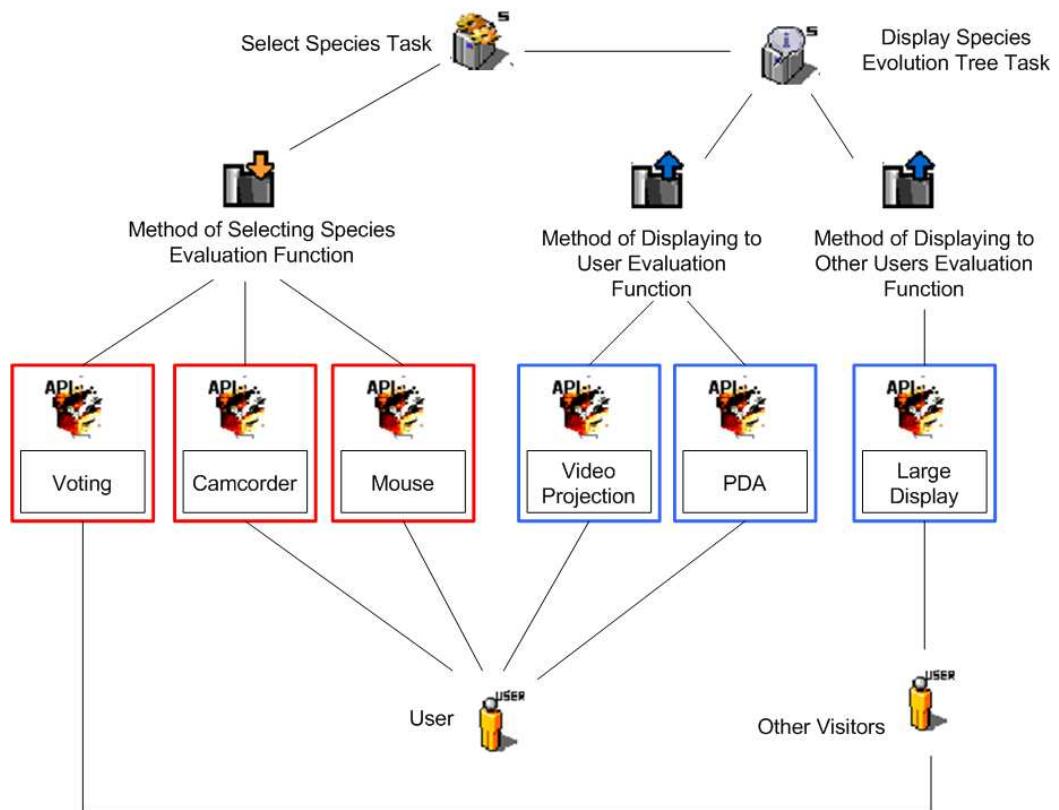
Figure 7.36: *ASUR/ASUR-IL transitional model with evaluation functions highlighted and selection of possibilities highlighted*

The ASUR approach currently enforces interaction design decisions to be made by developers rather than at run time by the users, but should the ASUR approach be adapted to allow for transformation from ASUR specifications to ASUR-IL at runtime then the approach described here, and pictured in Figure 7.36, could be used to allow for adaptable deployment of ASUR based systems.

7.5 Overview

This chapter has presented the work done on the MATCH project and was used to test the ideas in Section 5 in an actual implementation.

The framework implementation presented here uses a well considered design based on a publish/subscribe protocol and incorporates cutting edge design methodology in the areas

of task modelling. The design is highly modular and supports a variety of alternate implementations of infrastructure to suit the deployment target - specifically there exist multiple implementations of the Message Broker and Service Discovery subsystems that can be used alongside the evolution configuration related aspects of this framework. The ideas presented in Chapter 5 and Chapter 6 have been implemented within this framework in the form of the Interaction Manager and has shown that these ideas are feasible and can be used along side good development practise within the area of ubiquitous systems.

The framework was extensively profiled and shown that in the worst case performance it is still scalable enough to cope with the likely demands of a ubiquitous system within an office or home environment. This is particularly true of scalability in terms of number of components within the system where performance scales linearly and which is likely to be the primary metric governing scalability. The approach here works best for sparse graphs and may appear to be limited by this; however realistic dense graphs are actually very rarely encountered within real world systems [136].

The framework - and the model presented in this thesis - has been extensively used by many other people due to the ease of implementing advanced features with it. The framework has shown that the ideas in this thesis can be used to implement a wide variety of different applications. It has contributed directly to two Masters thesis, one PhD thesis and was extensively used as a case study in another project resulting in two peer reviewed publications. In addition, it has been used in a number of other student projects.

In the next chapter, two longitudinal investigations are conducted using applications built upon this framework.

8

Investigations into Evolution

> ## Published Work:
>
> This chapter incorporates material that has previously been published as *User Configuration of Activity Awareness* [146] and *Using Activity Awareness as a Runtime Interaction Configuration Testbed* [147].
>
> As first author my contribution to these two papers was developing the systems that they discuss as well as writing the majority of each paper in cooperation with my supervisor Phil Gray. The work in this chapter follows on from these publications.

This chapter sets out to address some of the research questions from Section 1.1.

Two longitudinal investigations were conducted using applications built using the software framework described in Chapter 7. The aim of the first investigation, in Section 8.3, was to study the processes, methods and approaches used while configuring a large or complex system in the context of a social network application and to evaluate the results in terms of the evolutionary process model presented in Chapter 4. The second investigation, in Section 8.4, was designed to build on these results to discover some of the relevant factors which affect the configuration process and subsequently the success or failure of particular configurations and to investigate it in the context of home care.

For both of these investigations, an interactive system was created with a large configuration

space within which these issues could be explored.

8.1   Activity Monitoring Technology Probes

These investigations were designed to allow participants to interact with other participants using a notion of Activity Monitoring; which allowed users to share activity messages with each other. Previous studies and stakeholder workshops conducted by Julia Clark and Marilyn McGee-Lennon as part of the MATCH project [36] questioned users on which types of technology they desired. Overwhelmingly, they most wanted to improve communication between themselves and their friends and family.

The activity monitoring system described here is a *technology probe* [110] designed to provoke a response from users as the result of using it. Technology probes balance the process of collecting information about users and their use of a technology with the engineering goal of testing and further developing the technologies being used in the probe. By contrast, the topic of activity monitoring is not intrinsically interesting for the purposes of this thesis, but is being used because it serves the technology probe role well.

Activity Monitoring provides a rich source of configuration; it is desirable for users to be able to discriminate which friends receive which activity notifications and what level of detail they receive. Activity information can come from numerous sources within the home or office; in addition to messages that the user can enter themselves via keyboard it could be possible to track the users' location or movement, monitor if they have taken medication on schedule, or if they have left their home. Determining which group of people is eligible to receive each activity message is not a trivial configuration task.

Activity exchange is a bidirectional communication channel; it is not only the person living in the home that is likely to generate messages. A grandparent may wish to know that their children and grandchildren are well. They will likely have multiple sources of messages from other people and they may attach different priorities to these messages; or wish to have them delivered in a variety of different ways. Activities from a distant friend may be added to a GUI presenting a list of received messages whilst a new message from a close relative may elicit more immediate attention with speech or audio notifications. However, these additional requirements make the configuration of an activity awareness application much more complicated as the user now needs to specify the precise relationship between activity message inputs (which may be automatic sensors, user controlled or sourced from other people) and outputs (which may be a range of devices or other people who should receive the message).

8.2 Analysis Methods

The investigations; given the aims described in Sections 8.3.1 and 8.4.1, and the constraints on how it could be performed (e.g. not many participants; difficult to control conditions; hard to collect data capable of being quantitatively compared and the need to be open-ended due to early stage of understanding of the domain) was best carried out via qualitative methods. There are a number of candidate approaches to the analysis of qualitative data [214]; three popular approaches which were considered for these studies are ethnography, grounded theory and framework analysis.

Ethnography [209] is one of the original methods of qualitative analysis common within the social sciences and elsewhere and many of its techniques are common to other approaches. Ethnographic studies aim to embody the nature of a person, culture, organisation or activity into a descriptive textual format through a variety of data collection approaches - such as participant observation, interviews or questionnaires - while the researcher immerses themselves within the culture and experiences of the person or thing being studied. Ethnography involves the observer participating in the life of the user for a prolonged period of time, both covertly and overtly, to gain an understanding of the behaviours under study. Ethnography was not feasible for the studies reported here given the limited access to participants and to the contexts of use.

Grounded theory [91] was developed to address the question of how to analyse data when there is no pre-existing theoretical foundation. Grounded theory formalises a process where, instead of initially developing a theory or hypothesis, the first step of research is immediate data collection followed by the drawing of inferences and conclusions based on the four stages of analysis listed below;

- **Coding**: Identifying codes which identify key parts of data

- **Concepts**: Grouping codes from the first step into groups or concepts

- **Categories**: Organising groups or hierarchies of similar concepts

- **Theory**: Devising theories that explain the subject of the research

A key point of this approach is that there are "no preconceived notions" [91]; it is believed that studying literature of the research area or formation of hypothesis causes preconceptions to be formed which will influence the results. As a result grounded theory is best suited to problem domains with interesting phenomenon without explanation which researchers wish to investigate and is unsuitable for the testing of hypotheses [215]. Grounded theory was not suitable for these investigations since it assumes that there are

no pre-existing themes or questions (all emerge from the analysis), while in this case some themes were dictated by the original thesis research questions.

Consequently, the approach used here was *Framework Analysis* approach, as described by Ritchie and Spencer [188], which attempts to capture the ability for both a priori and emerging themes to be analysed by incorporating techniques from both ethnography and grounded theory. The benefit of using framework analysis is that it provides systematic and visible stages to the analysis process. Although the general approach is inductive, this form of analysis allows for the inclusion of a priori as well as emergent concepts. This is important in human-computer interaction research because there are a priori issues, rooted in the design of the system and the underlying intention of the investigations, that should be explicitly addressed as well as unpredictable themes and issues that emerge during use in context.

Framework analysis involves the following five key stages:

- **familiarisation** - (immersion in the raw data by listening to recordings, reading transcripts etc.)

- **identifying a thematic framework** - (identifying the key issues, concepts and themes)

- **indexing** - (applying the thematic framework to the data)

- **charting** - (rearranging the data according to the thematic framework to create distilled summaries)

- **mapping and interpretation** - (understanding and finding associations between the themes with a view to providing explanations for the findings)

It is important to note that analysis does not take place in a linear form and that the five key stages overlap and feed into one another. In line with this the process of data analysis began at the point of data collection. Audio recordings were listened to several times in order to become familiar with the data and transcribed into a textual format for cross-referencing. A thematic (coding) framework was identified based on both a priori themes (identified as being themes of interest from the outset i.e. from the research questions identified) and emergent themes from the familiarization stage. Emerging themes were then identified and applied to the data to categorise and structure the data according to the themes. The final stage of analysis involved understanding and interpretation in relation to the identified themes which are presented as succinct design guidelines derived from the analysis.

As these investigations are qualitative in nature they are not designed to be representative in

terms of statistical generalisability and would gain little from expanded sample sizes except for more cumbersome datasets [178]. Instead they aim to allow an understanding of the experience and context of use of the participants.

8.3 Investigations into Evolutionary Configuration Processes

8.3.1 Evaluation Objectives

The aim of this evaluation was to investigate user behaviour when configuring multimodal activity awareness systems in the context of activity monitoring in an office environment. Specifically, the process for representing Interaction Evolution is presented in Chapter 4. Of particular interest was the extent to which this model is represented within user behaviour as well as the factors affecting decision making within each of the key stages of the interaction evolution model.

The focus during this investigation was the identification of broad requirements and behaviours from the participants and their attitudes as regards configuration approaches and interaction techniques rather than focusing on specific methods of configuration (such as manual or automatic approaches).

Additionally, this investigation was intended to show that it is possible to create very complex systems (in terms of number of available configurations) using the software framework described in Chapter 7 and that this framework is capable of evaluating the configurations in real time in response to user interaction.

To investigate these questions this application was deployed twice, once in the Department of Computing Science (DCS) in the University of Glasgow and again in Laboratoire D'Informatique de Grenoble (LIG) in France. This allowed a wider range of participants to be involved than if only one deployment was conducted.

8.3.2 Procedure

8.3.2.1 Participants

Participants were recruited from a self selecting group of Computing Scientists from the Department of Computing Science (DCS) (n=6, 5 male, 1 female) and the Laboratoire

D'Informatique de Grenoble (LIG) (n=8, 5 male, 3 female) for each respective investigation. Each deployment took place over a period of one week; the DCS investigation was conducted in December 2008 and the LIG investigation took place in January 2009. The investigation that took place in LIG was kindly funded by the Ken Browning travel scholarship program in the University of Glasgow.

Participants are labelled by groups using the patterns DCS*x* and LIG*x* (participant *x*) which represents an anonymised participant in the investigation. The notation *I:* within quotations is used to indicate questions from the interviewer. DCS indicates a member of the Department of Computing Science (DCS) in the University of Glasgow while LIG indicates a member of the Laboratoire D'Informatique de Grenoble (LIG).

Both of these user populations consisted of Computing Scientists within the fields of Human Computer Interaction - specifically both groups are interested the challenges of interaction with users with a particular emphasis on multimodal user interfaces, plasticity and context aware applications thereof. These participants were intentionally chosen due to their expertise in these areas and were sought out to provide expert critical feedback and criticism. In addition to the participants there was one user (DCSM - Phil Gray / my supervisor) within the DCS deployment, referred to in Section 8.3.3.10, who used the application but is not included in the overall results due to the fact that he was closely involved in the investigation.

The application was deployed within an office environment in both locations. Although all the participants were Computing Scientists there was a wide range of technical expertise within the investigation due to a large number of the participants being focused primarily on human centred studies.

8.3.2.2 Tasks & Context of Use

This investigation allowed participants to interact with each other using the supplied activity monitoring application. The application is capable of creating activity messages from a variety of sources which can be sent to a variety of destinations. Participants can share activity statuses with other participants and choose to receive their own, and other users statuses, on (among others) a graphical user interface, speech synthesis and email.

To do this participants must configure the application in order to specify *rules* or *configurations* which control the routing of messages within the application. Some configurations can only be achieved through cooperation; for example if you wish to receive messages from another participant that participant must choose to send you messages as

well and decide which ones to send.

Participants need to update these configurations as their situation changes and as they find improvements or flaws with their initial setup. The process by which they identify these opportunities for change and their behaviour in improving the system was the focus of this investigation.

The application was deployed in situ on ordinary desktop workstations equipped with USB Bluetooth adapters. Both deployments of the application occurred in academic office environments. All participants were friends with some subset of the other participants (i.e. they would all have people they could communicate with) although they were not guaranteed to know every other participant by name.

Both deployments lasted for a period of one week from beginning to end. Participants were given an introduction to the application of approximately one hour and an extensive user manual to explain the features available.

A detailed help guide and manual was provided for the participants, which contained detailed walkthroughs of the application with images along with summary images of relevant screens and features as provided in Appendix C.

8.3.2.3 Evaluation Platform

Figure 8.1 shows a high level overview of the architecture of this investigation which allowed each user to communicate with every other user. Each node in the graph represents a user in the deployment of the system. Users are capable of communicating with each other, represented by directed edges between users. Users can choose who they wish to exchange messages with.

A detailed exposition of the architecture, as used between two users, is shown below in Figure 8.2. Important concepts are that (i) the messages exchanged between users are not limited to textual messages and can be obtained from a variety of sources, and (ii) destinations for messages are not limited to other users. For example, a user could setup their own activity statuses to be displayed locally for their own purposes.

In Figure 8.2 the left-most vertical column for each user indicates input sources available to that user while the right-most vertical column indicates the destinations that each of the inputs can be sent to. The central column within each user in the Figure represents Monitoring tasks - in this application these tasks are composed of two evaluation functions: one to select the input sources and one to select the output destinations.
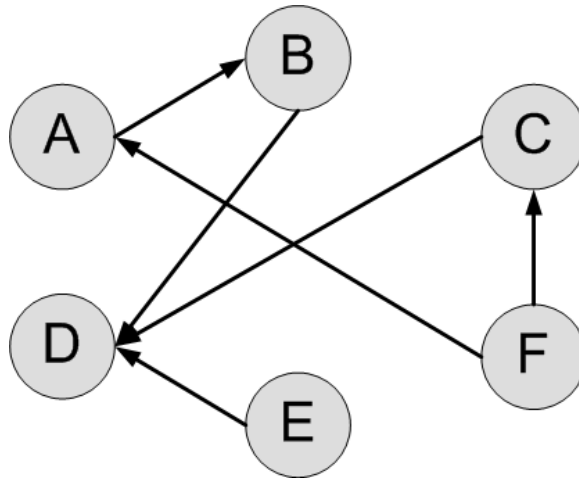
Figure 8.1: *Investigation Architecture*
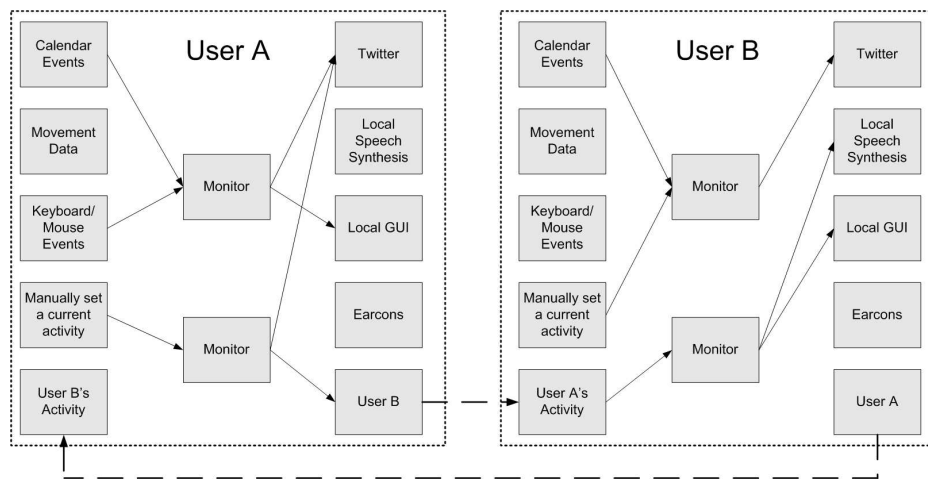
Figure 8.2: *Investigation Architecture - Detailed*

Even in this seemingly simple architecture diagram, there are a huge number of possible configurations. This can be shown by realisation that the number of possible interactions grows exponentially in terms of the number of inputs and outputs added. For example, adding an additional output device allows an additional number of configurations in proportion to the number of existing input devices already in the system. More specifically the number of unique configurations per user can be calculated as 2^{IO} (where I and O are the numbers of input and output sources respectively). This results in a very large number of potential configurations for a user to choose from.

This investigation included a wide selection of both input sources and output destinations.

Each of the input sources is capable of generating *activity messages* which can be directed towards a specified output destination. Sources may generate messages through implicit or explicit interaction with the user. The input sources available were:

- SHAKE
    - SHAKE/JAKE [233] devices are wireless accelerometer based interaction devices, similar to the Wiimote but physically much smaller. Activity messages are generated when the accelerometer detects movement (such as shaking) over a given threshold. SHAKE devices incorporated a push button which could be used to trigger activity messages.

- Webcam movement
    - A webcam monitored an area of the office for movement by comparing successive frames of video and generating a message when movement is detected.

- Idle time
    - Idle time measures the amount of time the desktop PC the application runs on has been idle, as measured by the last time the mouse or keyboard was used to interact with the machine. If the amount of idle time exceeds a user specified value then an activity message is generated to indicate this; successive messages are generated, with an rising period between each message, while the machine remains idle.

- Calendar
    - Activity messages can be generated based on reminders that have been set using a calendar application. This calendar application was built using Google Calendar[1] as it has an open and free API which allows applications to retrieve reminders. When a reminder event occurs with an associated reminder in the Calendar it is used as an activity message. Multiple reminders may be set up for one calendar event and each will trigger a separate activity message.

- Manually entered status messages
    - The participant can manually enter messages detailing their status or activities and categorise them as "Personal", "Work" or "Other". This allows different types of status message to be delivered to different people as each category is able to be independently selected.

---

[1]http://www.google.com/calendar

Similarly there were a variety of output devices which could be used to deliver activity messages either locally (directly to the participant using that instance of the application) or remotely (sending them to the application instances of other participants or via email or twitter). The output destinations used were:

- GUI

  - A Graphical User Interface shown in Figure 8.3 was present on the main panel of the application. This GUI was registered as an output and as such users could select which of their activity messages they wanted to be displayed graphically.

- Earcons

  - A selection of Earcon sounds were included (piano, marimba and clarinet) which were reused from Chapter 3. These were MIDI samples of different musical instruments with varying timbres and melodic ascensions. Each type of activity message had a different Earcon associated with it that would be played when the participant selected that activity source to be played as an Earcon.

- SHAKE vibration

  - The SHAKE device (described previously) has a integral vibrate functionality. This offers a subtle indication that a message has been received. Each type of activity message had a different pulse frequency.

- Speech

  - The Cerevoice [9] speech synthesiser was used in order to be able to output an activity message locally.

- Twitter

  - The Twitter[2] social networking application was integrated such that it could be used as a destination for activity messages. These were posted to a user's account so that existing Twitter users who followed the application user could retrieve activity messages posted in this way through the regular Twitter interface.

- Email

  - Messages could be sent to user selected email addresses.

In addition to the inputs and outputs specified above, the application modelled other users as both message sources and destinations. In terms of component modelling, the

---

[2]http://www.twitter.com

other participants in the investigation were selectable as destinations or sources of activity messages as shown in Figure 8.2.

Figure 8.3 illustrates the main interface used on the desktop. The application was designed to use a small amount of screen real estate during phases when the user was not actively configuring it. This resulted in a small application window that could be left in a small corner of the screen or minimised while running, depending on the user's preferences.
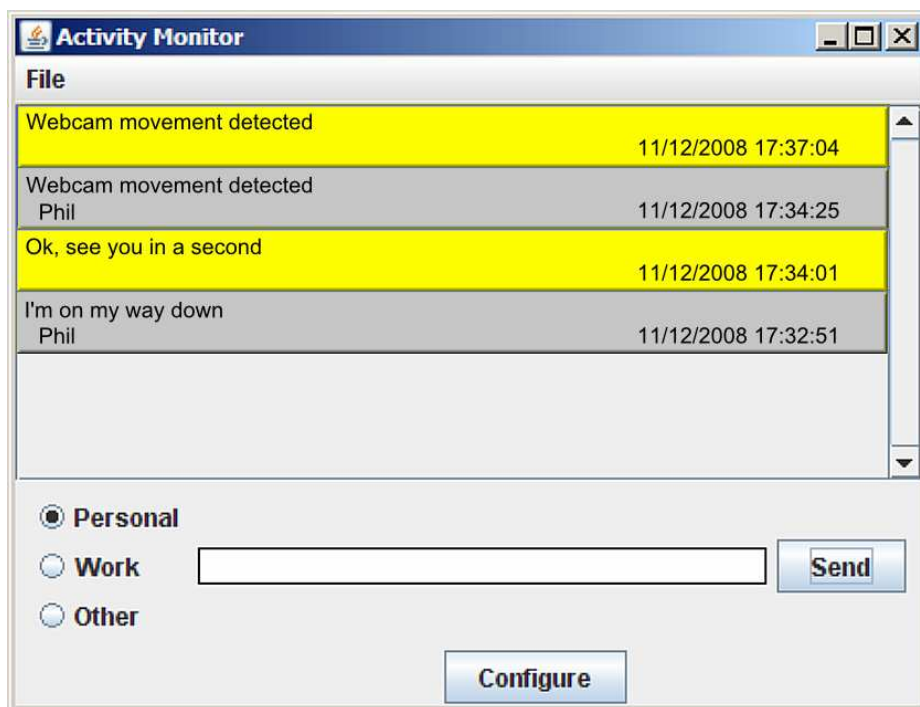
Figure 8.3: *DCS Interface*

Figure 8.3 shows the components of the monitoring interface; the GUI message window, with a single example message in yellow and the "enter an activity" text entry fields. The former section allows space for graphical display of messages you have sent and received (when configured to do so) while the latter section allows text messages to be composed and sent. The other input and output devices do not require permanent screen real-estate to operate.

There is a *Configure* button at the bottom of the main interface; when selected it will display the panel shown in Figure 8.4 which allows configuration of the application to take place.

This configuration window allows monitoring tasks to be added and removed as well as setting up the input and output sources for each task (the left and right hand sides of the panel respectively). The application provides a number of approaches to configuring input
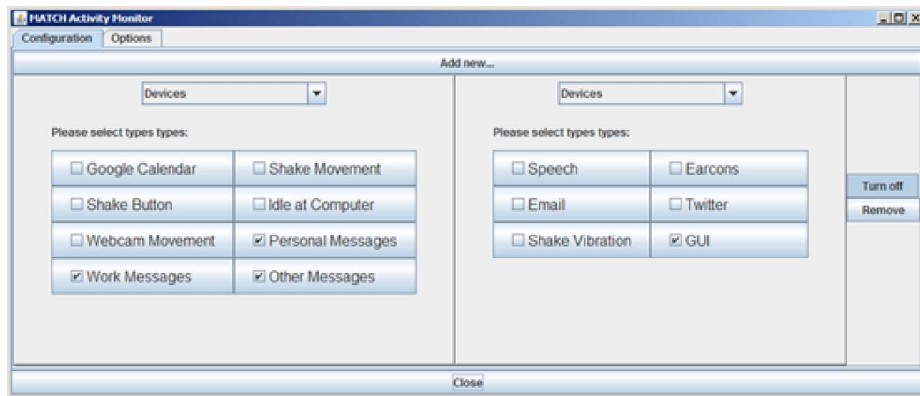
Figure 8.4: *DCS Configuration*

sources and output destinations, which are selected from the drop down menu (one for each input and output).

A number of manual interaction approaches were provided; **Devices**, **Groups of Devices (Preselected)**, **People**, **Groups of People (Preselected groups)** and **Groups of People (User Selected) - LIG only**. In these approaches, the devices or other users are selected by the user manually choosing the desired devices. The approaches are categorised further into "Individual" and "Groups of..." approaches; users can either pick the specific device or person themselves or instead opt to select a group of devices or people.

The next two approaches are fully automatic approaches (**Standalone Recommender** and **Collaborative Recommender**) which will select appropriate choices for an input or output automatically based on local or collaborative usage history. In addition to the automatic recommender approaches there was a **Manual Recommender - LIG only** which would automatically suggest appropriate configurations but would rely on the participant to make the final decision.

Furthermore, there are **Combine two other options** and **Context Sensitive** functions. These two approaches allow for multiple approaches to be combined on one monitor task. For example the combinatory approach takes the union of two other approaches in order to combine their results - allowing the use of more than one function within a single monitoring task. The Context Sensitive function enables a user to switch between two other options based on some condition; in the application used for the study, the only condition available was a user's machine being idle for a specified period.

The **Nothing** function can be used to clear an existing task or within the Context Sensitive function when the user wants nothing to happen when you are at or away from the machine.

As listed in this section, there were a small number of functional differences between the DCS and LIG investigations. The DCS investigation used the SHAKE devices shown in Figure 8.5 while the LIG investigation used the newer JAKE devices shown in Figure 8.6. The JAKE device is smaller and has a longer battery life but removes the vibration and push button features.
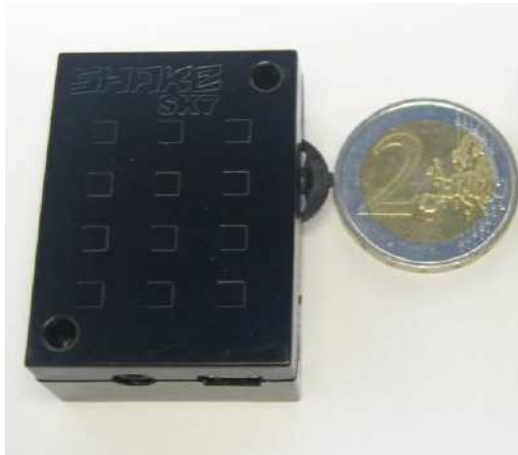
Figure 8.5: *SHAKE device by SAMH Engineering*

Figure 8.6: *JAKE device by SAMH Engineering*

An option was made available to run the application without a webcam for participants who did not wish to use the webcam or who already currently used a webcam extensively for Skype or similar services and as such could not afford to allow this application to take exclusive control of the device.

Two of the evaluation functions described were developed during the time between the two deployments (listed as LIG only). These additional evaluation functions were (i) a function which allowed he ability for participants to create their own groups of people rather than using preselected groupings, and, (ii) a recommender system that allowed a much greater aspect of control by allowing the participant to manually select the desired configuration from a ranked list generated by the recommender instead of automatically selecting the highest ranked possibilities without user input.

8.3.3 Results

The focus of this investigation was to investigate via interviews, and analysis of accompanying log files, the different approaches that participants took in order to create, change and remove rules from the system as well as the processes involved in terms of how

users identified unsuitable configurations. It was found that each of the four key stages of configuration identified previously, in Chapter 4, was identified as an emerging theme within the interview data gathered. The following sections discuss each stage in more detail along with supporting themes and comments from participants.

Following the framework analysis a number of themes were identified including the a priori themes identified in Chapter 4. To recap, these are:

- Identification of opportunities for change

- Reflection on alternatives

- Decision making

- Implementation

Five emerging themes were identified and were common across both investigations, namely:

- Methods of Configuration

- Approach to Configuration

- Usage of Activity Monitor Application

- Interaction with other Participants

- Control and Understanding

Each of these themes will be discussed in turn using stakeholder comments to support and illustrate each point and an analysis of participants messaging behaviour is presented.

8.3.3.1 Identification of opportunities for change

The primary cause of reconfiguration was the discovery of unintended consequences, such as a rule executing at an inappropriate time or a rule failing to achieve the desired affect. This could arise from, for example, the connection of a high frequency or high sensitivity data source to an unsuitable modality, such as speech synthesis or earcons.

> LIG3: "I saw it immediately because it didn't have feedback on my screen."
> DCS6: "I had just left my speakers plugged in like and say I had like a meeting, a couple of people came in for a meeting and then all of a sudden speech started going mad."

Other times the rule set would change as the user grew more accustomed to using the application over the course of the investigation and discovered that the application either worked differently to their original assumptions or they realised additions or alterations they could make to improve their configuration.

> LIG2: "In fact during the week, I just begin with setting up some rules which then I had to change them - because I was realising it was working a little bit differently and I needed something else."

Another reason for frequent reconfiguration was the change of circumstances over the course of the day which would require reconfiguration in order to satisfy the new requirements.

> LIG2: "In fact my needs were different depending on the time I was using."

A final stimulus for change was the usage of test rules in order to experiment before actually connecting the rule to its intended outputs. An example of this was when participants would experiment with a configuration but intentionally limit the scope of the rule in order to examine the behaviour of the rule to ensure it will do what is intended before revisiting it at a later time when the rule is then adjusted to fulfil the original goal.

Guideline/Observation 1. *Stimulus for change can come from a variety of sources, both internal and external to the application. (Current configuration found to be unsuitable, gains in understanding or experience prompting change as features more fully understood, change in circumstances making earlier configurations now unsuitable and experimentation with possible new configurations).*

Guideline/Observation 2. *Stimulus for change can result from both planned actions (experimental evolutionary changes) and unplanned events (response to a change in conditions).*

Due to the nature of the different communication configurations, some configurations are likely to be fairly stable (peripheral information to a group) while others may be very volatile (specific message to a specific person). This application uses a single unified interaction technique for all and as such the cost/benefit ratio may change for different configuration changes. The optimal approach may be to provide complementary configuration interfaces for the different purposes (e.g. quick configurations for sending individual messages).

8.3.3.2 Reflection on alternatives

When reflecting on alternatives participants employed an exploratory approach. The extreme form of this was users who would create a rule without any prior knowledge of what they wanted the rule to do but would explore until they found interesting features of the application that they wanted to include in the rule.

> DCS3: "I looked at the configuration screen and thought: that might be interesting - click."

A variation of the exploratory approach was the exhaustive search where a user would experiment with each of the available options and eliminate the ones they did not want to use until they were left with the most preferred option.

> DCS4: "Well there was a list of options and I just went through them one by one and just took them and discarded the ones I didn't want."

This is a very similar mechanism to the exploratory approach but represents a filtering approach to the task rather than the hunting and exploratory approaches previously described. This suggests that there must be a variety of exploration methods made available to users which can support a mix of different approaches, including directed and exhaustive searches.

Guideline/Observation 3. *Users must be able to explore alternative configurations, both iteratively and by searching for specific or interesting features.*

Allowing the participants to merely consult the rules they had previously specified did not allow them to reflect properly on the alternatives, as it could be difficult to imagine the consequences of rule changes as the effect of a rule change may not have an immediate effect.

> LIG8: "So there is one part for the rules and another part for the messages and when I was typing my message I didn't know which rule was applicating."

Guideline/Observation 4. *The application of multiple - potentially overlapping - rules can result in difficulty determining the current behaviour. Facilities to provide an overview of the current configuration may be helpful to users.*

8.3.3.3 Decision Making

Decision making for the participants was often conducted in light of previous experience with the application and their opinion of how good a particular previous configuration was.

> DCS6: "So quite quickly the event that made me make my decision was several million speech things coming through which started to get annoying."

Decision making at the beginning of the investigation was therefore much less informed and much more prone to configuration of rules which were eventually found to be unsuitable.

> DCS2: "I'd say, it was a bit over time, so I'd choose it went to the web browser, [then] went to do my work and came back and it totally wasn't what I thought and it wouldnt do for me."

This was explained as being a learned process centred around determining what they actually wanted to use the application for.

> DCS6: "To start off with there was a lot of sussing out about; like right, What do I want to do? and What can I test it?, What's sensible to test it?"

Guideline/Observation 5. *Users should not be tied into decisions they make early on as their decision making is least informed at that point in time.*

Decisions involving reconfiguration do not necessarily take place entirely within the application system - for example, in one case, DCS6 simply decided to unplug the speakers in order to silence the effect of a rule which was causing audio alerts during an impromptu meeting.

This may indicate that unplugging the speaker was easier than reconfiguring the rule; this may be because any configuration - no matter how small - requires some amount of mental effort to identify the rule at fault and to correct or disable it. Alternatively, unplugging the speakers represented only a small physical effort and prevented attention from being diverted from the more important meeting at hand.

Of course this has consequences as the rule is still active and when the speakers are reattached the existing rule will still deliver audio notifications - possibly as a surprise to the user as well as disabling other audio notifications that may be delivered from other applications.

Guideline/Observation 6. *The overall context of the user or computer can be changed without changing the system configuration - if the system were to try adapt to this change it may actually defeat the user's wishes.*

8.3.3.4 Configuration Implementation

Rule creation was dominated by users who would create a number of very simple and complementary rules in order to implement their requirements.

> DCS2: "I think I tried to and then thought instead of combining them - just set up individual rules and just went down that route."

The function of some rules may be superseded by the addition of new rules at some later timepoint. An example of this is when a message is sent to a subset of people but then a more general rule is created which can also perform the original function. As a result these rules can be said to be overlapping - detecting these overlapping rules is not addressed in this thesis but is discussed in Section 9.3.1 as a potential form of future work.

> LIG1: "For example, I created a rule to send messages only to LIG5 and then, when I realised that it was working for LIG5, I wanted to send messages to all participants, so I already had this rule, so I deleted the one for LIG5."

Guideline/Observation 7. *Users tend to create collections of small, cooperating, rules as part of a configuration.*

Although participants would create a number of rules they would typically focus on one rule at a time before moving onto the next - ensuring it was satisfactory (at least in the short term) before attempting to tackle other rules.

> DCS3: "I tried one rule at a time and it was very much just messing about with that just to see what happened."
>
> DCS2: "It was easier to see, in my opinion, it was easier to see each individual thing (rule) rather than trying to combine things - I didn't want to overcomplicate things in the way of - I'm gonna sit and tag three or four things and then try and work out - It's gonna do that and that."

Guideline/Observation 8. *During rule creation, users prefer that the emphasis is placed on a single rule at a time instead of trying to display the overall behaviour of the application. This may, however, interfere with a users requirement to be able to detect interacting rules.*

Participants opinions were split on the preferred interaction technique used to set up the rules. One highly technical user suggested a preference towards usage of policies expressed in natural language tools in order to specify rules such as Wang et al. [229].

> LIG6: "Ah, mostly something, look like more something some kind of natural language, uh a simple one that some kind of."

This might be due to this particular technical user being familiar with functional composition languages in the past as most other users expressed precisely the opposite preference.

> DCS6: "I don't think I'd have liked some sort of, em, command based language just to set up the rules, even thought that would have been a lot quicker if you knew what you were doing."

Clearly the type of specification languages made available should be designed around the expected technical capabilities of the user.

Guideline/Observation 9. *Approaches for specification of rules depend on users' technical ability and previous exposure to an approach and should be targeted accordingly.*

However, even proficient users still learn how to configure the system more effectively over time on their own. Even though they are capable of manipulating the rules and configuring the system their knowledge of how to configure a system and what to use it for evolves as they learn how they can integrate this with their daily activities in order to take advantage of its functionality; this process appears to be ongoing.

> DCS6: "It's amazing how much time you actually need, not just to set up the rules, just to get your head around the best way to use the rules for you. So I think more time would have revealed, you know, what way I would have probably set things up longer term."

Guideline/Observation 10. *Users will gain experience and knowledge for effective implementations and configurations over the course of time using the application as well as via interactions with other users.*

Conversely, a much smaller group of people did not use this approach, and in fact would not adjust rules at all after creating them.

> DCS4: "Well, I only configured it once."

There could be many reasons for this behaviour; they may have not found any strong uses for the application, they may not have interacted with the application enough to build up experience and knowledge for effective implementations or they may have set up an adequate configuration from the start and seen no need to change it.

8.3.3.5 Iteration

This section explores the different approaches that participants took in order to create, change and remove rules from the system as well as the processes involved in terms of

how users identified configurations that were unsuitable.

Evolutionary approaches to identification of opportunities, reflection upon possibilities and to decision making and implementation were previously discussed in this chapter but it was found that the overall approach was typically evolutionary and exploratory in nature as well resulting in an iterative approach among many of the participants. This was demonstrated by users who would continually refine rules.

> DCS6: "I was kinda like building up from straightforward rules to more complicated; I was trying to kind of like investigate all the different things that I could do."
> LIG4: "[I] created a few rules, a couple of rules, and er, [I], er, enriched them."
> DCS2: "Just kind of played around with it and you know, kinda refined it."

Guideline/Observation 11. *Exploratory and evolutionary changes are frequently made in order to refine a configuration over time.*

However, sometimes this exploratory behaviour could be stifled by a type of loss aversion where there was an unwillingness to change something in case it resulted in a configuration that was less optimal than what they already had.

> DCS1: "I thought it might not work so I just kept it as it was."

Guideline/Observation 12. *Loss aversion can inhibit exploratory and evolutionary changes - it should be possible to restore a previous configuration to limit this aversion.*

Disabling or changing portions of a configuration encourages evolutionary configuration changes - that is, where a participant can experiment with portions of a configuration at a time to determine what they actually want to happen and to try out alternate configurations.

> DCS6: "I got lots of testing going on... once I was happy that it was definitely going to the speech, and going to the earcons, and going to something else - I'd then say, like, which, what do I actually want it to do for long term and then I'd like deselect certain bits and say 'Right, I only really want that to go these two things'."

Guideline/Observation 13. *The ability to disable or alter rule configurations without destroying the previous state encourages evolutionary behaviour.*

In addition to evolutionary changes there was a type of change that was purely reactive in nature in order to correct a problem that had become immediately apparent. In this change

the aim of the reconfiguration is usually very specific and is set out to tackle one particular issue that is causing them problems.

> DCS3: "Turns out if it's one minute idle, then yes, it gets a wee [little] bit annoying with earcons ... can feel your vein throb in your temple.... I never took earcons off the idle but it was far more manageable once I set the time limit that I was idle to for a longer period of time."

> DCS6: "I had just left my speakers plugged in like and say I had like a meeting, a couple of people came in for a meeting and then all of a sudden speech started going mad."

This type of incident would usually prompt immediate reconfigurations.

> DCS2: "Yeah, once I realised it - it was just like 'take that off'."

Reactive changes are typically symptoms of problems which have been identified with the current configuration and will often aim to disable or cancel some portion of a configuration. This implies a need for being able to quickly disable "rogue" configurations which have been identified as being a problem for the user.

Participants demonstrated that they could predict the success of a configuration in advance once they had become proficient with the application and knew which options were available to them and were more confident in making immediate changes with a known outcome.

> LIG2: "I knew which rule would work, so I just chose the right one and that was it."

Guideline/Observation 14. *In situations with unchanging context or with context changes that are well understood, advanced or proficient users are capable of identifying the configuration required without evolutionary experimentation or outside assistance. Unnecessary assistance may impede their configuration rather than helping it.*

8.3.3.6 Methods of Configuration

The application allowed participants to select devices and people using a variety of different approaches and at different granularities. Participants strongly preferred approaches which allowed individual and specific selection of devices rather than selecting using the more abstract "Groups" functions. The stated reasons for this were preferences towards approaches which offered more control.

DCS6: "I think I pretty much always used people or devices [As opposed to groups of people, groups of devices]."

DCS1: "I suppose it was a bit more precise, so you know, know better what it's doing. [Referring to selection of devices individually]"

This may be because devices have a number of non-functional attributes which may influence selection - an accelerometer based movement detector is not equivalent to a webcam based movement detector in function - and as such a greater level of control was desired in order to make the correct task specific choice. This preference was not as pronounced when selecting between people and participants were more comfortable using groups in this situation.

DCS2: "It's just people and then groups of people. I did like the groups of people thing because it's easier."

This may be because people are more familiar with the concept of forming personal and social relationships into groups where each member of the group is treated the same but this requirement may not not satisfied in groups of devices.

Guideline/Observation 15. *Not all configuration components can be treated equally. Non-functional criteria can affect the optimal approach for configuration and developers should target selection mechanisms appropriately based on the items to be selected.*

Use of groups for selecting people was not ubiquitous and some participants still strongly preferred to select each recipient of a message individually even if a pre-existing group exactly matches their desires. For example in the following comment, the participant explains how they acknowledged the existence of an 'Everybody' group but chose not to employ it.

LIG4: "[I] wanted to make sure [I] can send messages to, eh, everybody because when you choose a group sometimes you miss someone in the group."

One observation is that this process of individual selection leads to a greater sense of control over the application which may not be satisfied when selecting categories or groups of devices or people.

Guideline/Observation 16. *Optimal selection mechanisms can vary between users. Some users value control over convenience while others prefer the tradeoff in the other direction.*

Participants in the DCS investigation suggested a recommender which, instead of automatically selecting the best possibilities, would instead provide a list of the recommended

205

possibilities and allow manual selection instead. The automatic recommender was not well used.

> DCS6: "So it would have been nice to maybe have a set of recommendations or you know popular choices... you know this is what people have used before and said works or whatever."

To address this, a modified form of recommender was introduced into the LIG investigation alongside the automated forms which would rank the available options for the user and present them sorted by 'score'.

However, this was also rarely used and the reasons provided for not using either the fully automatic or the manual selection form of the recommender approach typically indicated a lack of time or difficulty in understanding the point of recommenders.

> LIG8: "I don't have time in fact."
>
> LIG1: "I, I saw it - but, eh, finally I don't use, why I didn't I don't know."

Guideline/Observation 17. *Recommenders should not be considered as an absolute replacement to manual approaches and may have higher mental overheads.*

It was possible to combine two approaches within a single rule to make a more comprehensive rule, for example to allow selection of people, devices and the use of recommenders within a single rule. Realisation that this feature was available principally occurred after a period of use of the tool and when the participant realised that it was required functionality to meet their needs.

> LIG5: "It took me some time to understood that it was possible to combine several effect in one rule."
>
> DCS6: "I remember at the time thinking... I need, I can't just do a simple rule I need to combine them with something and when I quickly realised that I could do combination I was like - oh right, I can totally do a combination that's perfect."

Guideline/Observation 18. *Features may be ignored entirely until a situation develops that requires them. They can then be discovered when the user searches for them.*

The Context Sensitive function allows a particular input or output option to be selected after a computer has been idle for a specific duration. In both studies, this feature was used exclusively to switch between different message output modalities.

> LIG1: "I, er, I try it but I didn't send this information to the others, I just display it on my own."

LIG4: "Eh, it is very useful when you are not in front of the computer and someone tries to communicate with you."

There were a number of users who disliked or found no use for contextual switches.

LIG5: "Not something I wanted to use."

LIG3: "LIG2 explained it to me, but I didn't really need it."

It was surprising to find a user who actively opposed using context within the application since such a large proportion of ubiquitous system focus on context awareness as a primary goal. The comments from DCS4 below indicate a preference for a single state system which does not adapt based on context.

DCS4: "The way how I think of the system is when I'm actually at the computer and telling other people what I'm doing, the concept external events happening while I'm not near it is not of interest to me... because in my context there is only one context and I'm at the computer."

This was interesting as current literature often assumes users want adaptive systems when in fact some would rather a static system that does not change - but which may still need to be highly configurable.

Guideline/Observation 19. *When contextual information is used it is primarily to control output devices.*

Guideline/Observation 20. *Not all users consider contextual adaptation as being useful and prefer to respond to contextual changes manually.*

This may be the result of the user primarily viewing the application as a messaging system rather than as an activity monitoring system. In the former case there is potentially less value in context changes.

8.3.3.7 Usage of Activity Monitor Application

The primary usage of the system was split between participants who employed a people-centric view of the system and those who formulated a device-centric view.

DCS6: "I think because the way I was thinking about the rules ... I was thinking about them two ways. I was thinking - who do I want to know about this thing or who do I want to let know about .. so it would all start from that and then worry about - well how do I want them worry about the device or the

207

output modality then. So more often than not I think I was thinking people and then worrying about the device."

LIG6: "I liked playing with [an] object in my hands so it's the device that attracted me more."

Guideline/Observation 21. *There can be multiple, alternate, mental models in use by users which can affect their focus when configuring an application. User interfaces with different foci may be more suited for one type of model than another.*

Aside from the desire to notify other participants about their current status a number of participants used the facilities to notify themselves of their current activities. A typical example was appropriating the measure of idleness and using it as a self-correcting monitor to ensure that they were maintaining a desired rate of work through the course of the day.

DCS3: "Like you know if you'd been less industrious - it would let you know when you'd been away from the interface for five minutes and maybe its time to start getting back to work now."

Guideline/Observation 22. *Appropriation of the system features is a common use case in order to address needs not originally envisioned by the developer.*

Appropriation has been observed previously [58, 65, 162] in terms of Computer-Supported Cooperative Work (CSCW) and the findings here serve to confirm the same behaviours within the configuration and activity monitoring domains.

It was observed that audio based interaction devices are typically used for infrequent events. Several users reported beginning the investigation with intentions to use speech to allow themselves to be interrupted.

DCS6: "It might be kinda good so that the speech would interrupt me when, cause speech is good enough that it would have interrupted whatever I was doing."

However, both of the audio forms of alert (earcons and speech) were used as supplementary notifications of important messages.

DCS3: "But the earcons, and even when I had headphones in, so you hear through the headphones even when I've got them off, so it's a wee [little] pager."

Guideline/Observation 23. *Audio is typically used only for important messages designed to interrupt the user.*

When users were already familiar with Twitter services they were quick to adopt the new method of directing activity messages but those users who were unfamiliar with the service before the commencement of the investigation would tend to avoid including them in their configurations.

LIG6: "I don't use Facebook or Twitter usually."

Guideline/Observation 24. *Previous usage of social networking sites was a factor in uptake of the Twitter notification method. Lack of familiarity served to discourage users from using it.*

The GUI based notification features were typically used for messages they wished to receive frequently or to refer to later on. However. users found them inadequate for situations in which they were not directly looking at the screen.

DCS3: "When I was idle I was most likely not looking at the screen so the GUI one wasn't particularly useful."

Many of the participants reported enjoyment from interacting with the accelerometer based devices and found them to be an entertaining method of signalling other users. This result indicates that tactile based interaction devices can be a good method of encouraging participation within longitudinal investigations as they appeal to participants.

DCS2: "It was just DCS6 would shake the shake [laughs]."
LIG6: "I liked playing with [the] JAKE."

Guideline/Observation 25. *Tactile devices, through novelty or otherwise, can be enjoyable for users in terms of interacting with other participants.*

There were a number of distinct messaging strategies observed during this investigation. The first strategy observed was the broadcast approach to messaging other users. This would typically manifest as selecting all available participants to send messages to regardless of existing social relationships. This approach was frequently used when the user selected who and where they wished to receive messages from - often participants would select the broadest available categories for receipt of message. This is often to prevent them from missing any messages that may have been directed to them. This behaviour is reminiscent of social network applications, such as Facebook[3], which delivers messages collectively to a large number of interested friends.

The other common approach was to target messages to specific persons in order to target messages more narrowly such as might be done with an email application.

---

[3]http://www.facebook.com, accessed 2010

There were combinations of these approaches where a user would select to receive messages from everyone but only to post them to a limited number of preselected people based on their existing social relationship.

A number of users from both investigations (DCS4, DCS5, LIG6 and LIG7) reported using the application primarily to publish activity reports but not to receive them. This was confirmed via log files of the user's actions that at no point did DCS5 or LIG6 ever configure the application to receive messages from other participants - instead using the application solely to notify other participants of their status.

> LIG6: "Just to say - I am here."

Analysis of log files of interaction between users indicate that there is significant under-reporting in terms of users who report that they configure the system to receive messages from other users - instead reporting mostly configurations in which they were sender of messages. This is discussed further in Section 8.3.3.10.

8.3.3.8   Interaction with other Participants

The participants were asked to describe occasions on which they spoke to other participants about using the system, about the contents of the messages they sent over the system or about the configurations they had set up on the system.

Interaction outwith, but concerning, the system would typically be either testing, discussion of message topics or would concern configuration of the application. For example testing the application with another participant.

> DCS2: It literally was like DCS1 tried to set it up so he could send messages out and was like 'did you see that message?' and I'm like no and then we'd try it again."

Some participants agreed on common codes to use while using the system.

> LIG3: "And for the check box, I uh, discussed with LIG2, it was when we activated the checkbox it was a synonym of a break, it was a break, we were in a break."

In the above example, the two participants agreed that a particular feature of the system which was easily activated and sent to the other users would signify that they were going for coffee without the overhead of actually writing the message. This is reminiscent of behaviour observed by Richie and Mackay [187] who deployed an shared clock which allowed users to leave abstract markings for each other and who ultimately developed a

coding system for the abstract markers.

Guideline/Observation 26. *Users are capable of constructing abstract coding systems which are implemented on top of application functionality to meet their communication needs.*

Another frequently reported discussion point related to the configuration of the system itself. Often this occurred during the familiarisation phase of using the software when a more experienced user would assist with the transition from beginner to proficient user. The participants would frequently assist each other with configuration tasks and discuss the implications of different configurations.

> LIG1: "Yeah, the first time I created a rule, I said I was with LIG2, he showed me how it was, eh, done, eh. The second time it was with LIG5 on the phone and after I taught myself."

These education based discussions involved sharing of configurations between users in order to duplicate results.

> DCS6: "What does he have set up? And then I can pick and choose whether I want to, like, copy those configurations because to start off with there was a lot of sussing out about, like, right what do I want to do and what can I test it, what's sensible to test it... I didn't have a clue at first... for the first half hour."

However, these discussions were not always instruction based and often included descriptions of what they could use the system to do rather than specific instructions on how to do it.

> LIG2: "Yeah, we discussed it but in a general way. That is, I never really told the other people what I was presently doing but I told them what it was possible to do."

Guideline/Observation 27. *Users will share and discuss ideas for configuration in an ad hoc fashion. Often these meetings will include explicit sharing of configurations in order to help each other and general advice or ideas.*

Surprisingly, some people reported not interacting with other members of the study at all.

> DCS1: "[laughs] Just set it yeah up and hoped they did the same."

However, it was often the case that another participant would report communicating with that participant regarding the system.

DCS2: "It literally was like DCS1 tried to set it up so he could send messages out and was like 'did you see that message?' and I'm like no and then we'd try it again."

Guideline/Observation 28. *Not all interaction between participants will be reported during debriefing.*

8.3.3.9 Control and Transparency

Control and transparency of an application are tightly interwoven with each other. For a sense of control to exist it must be possible for the user to observe that interactions with the application have the intended outcome.

The approach of allowing the participants to specify rules and then to consult the rules they had specified did not offer this feeling of control to the users as the rule change did not always trigger an immediate reaction in the system (for example rules which notify about messages from other participants would only demonstrate the new behaviour upon the next message from another participant).

LIG2: "It was hard for me to see as a whole the behaviour of the system - so, before sending a message, and most of the time, I was going back to the, monitoring system to see how i had configurated it and to remember just how it worked."

Much of this was down to the observability of the different rules and their application. Although the participants found themselves adept at creating and modifying rules they found it much harder to maintain a coherent state of the system when multiple rules were applied at the same time.

DCS4: "I'd have felt more in control if I could see the results of my actions going to other people."

Guideline/Observation 29. *Users value a high degree of control and transparency in configuration approaches.*

This was combined with uneasiness with potentially sending messages or sharing information with people accidentally because they had misconfigured a rule.

DCS5: "Just sort of uncomfortableness about who was watching me you know."

This is evidence that even while a user may be comfortable and able to configure a system that this is not necessarily sufficient for them to have a full understanding of the system capabilities.

It may be possible to address this issue by augmenting the feedback the system gives by displaying the current and possible configurations of the system in a manner that reflects the system's capabilities. This is particularly relevant in respect to the research questions in this thesis; viz. what is the system currently doing? One approach to addressing this might be to include a confirmation of the current behaviour upon the change of the rule in order to reinforce that such a rule change had taken place. This might take the form of status or overview screens to show the current configuration or in the simplest case confirmation dialogs to confirm the new rule has been applied along with a mechanism to receive an overview which rules are currently in effect.

Guideline/Observation 30. *It should be possible to view the current (and alternate) configurations as an overview separate from the rule definition.*

This was reinforced by some of the comments received during the DCS investigation regarding approaches which would automatically select appropriate devices for you. If you specified an input or output then it would automatically select its generated recommendation for the complimentary portion of the rule.

> DCS5: "I was aware of them but I didn't really understand them, if you know what I mean, I didn't really have an idea in my mind of what they were going to suggest to me..." "I don't think id have wanted the software to be completely openly deciding what input devices or whatever it output devices to use - I definitely did want a bit of control."

Trust and the level of control were important issues that prevented the uptake of recommendation-based approaches to configuration. However, there were a variety of reasons provided for not using either the fully automatic or the manual selection form of the recommender approach.

> LIG2: "There are a couple of things that I never used... recommenders... I was rather do the things myself."
>
> LIG3: "I, uh, didn't use it because I didn't quite understand what it was about."
>
> LIG4: "[I] didn't trust eh, the, this device so [I] prefer to, uh, make the setup himself by myself."

Guideline/Observation 31. *Recommenders may have lower levels of trust or deprive the*

user of a feeling of control.

The preference for not using combinatory or contextual features was suspected to be a result of the low level of control that participants experienced when configuring a rule using these techniques and may be a result of additional complexity that may have been imposed when creating larger rules.

> DCS2: "I wouldn't say it was too complicated - I think it was just easier for me to kind of, I felt I had - I know this sounds silly - but I felt I had more control having individual things rather than having to combine two options together in one rule."

Guideline/Observation 32. *Lack of knowledge about the global state of the application can cause the user to feel less in control and less confident about their use of the system.*

8.3.3.10   Messaging Behaviour

During this investigation the application logged configuration changes to allow for later analysis should participants be unsure about the configurations they had made. During the post-investigation interview stage of this study, participants were asked to identify the users they set up configurations with during the study. Users were specifically asked who they wanted to receive messages from rather than whom they actually received messages from as we were most interested in configuration of the application rather than the social questions of who talked to who.

Figures 8.7 and 8.8 show the reported communications between users. Each user's reports are represented by a horizontal row with each column representing another user who they could have communicated with. Cells containing an S indicate the participant configured a rule to allow Sending of messages to another participant, while cells containing an R indicate the equivalent rule to receive messages. A cell containing S/R indicates that rules were set up for both sending and receiving messages. In the DCS reports the row for user DCSM is omitted as they were not a bona-fide participant (see Section 8.3).

Observations from looking at Figures 8.7 and 8.8 are that very few users report setting up configurations which allow for the receipt of messages or intending to do so. There are a large number of blank cells where no configuration was reported.

	DCS1	DCS2	DCS3	DCS4	DCS5	DCS6	DCSM
DCS1		S/R	R	R	R	S/R	S/R
DCS2	S/R		S	S	S	S/R	S
DCS3	-	-		S	-	-	-
DCS4	S	S	S		-	S	-
DCS5	-	-	-	-		-	R
DCS6	S	S	S	-	S		R

Figure 8.7: *DCS Reported Communication*

	LIG1	LIG2	LIG3	LIG4	LIG5	LIG6	LIG7	LIG8
LIG1		S	S	S	S	S	S	S
LIG2	S		S	S	S	-	S	-
LIG3	S	S		S	S	S	S	S
LIG4	S/R	S/R	-		-	-	-	-
LIG5	S/R	S/R	S/R	S/R		S/R	S/R	S/R
LIG6	S	S	S	S	S		S	S
LIG7	-	-	-	-	-	-		-
LIG8	-	-	-	-	-	-	-	

Figure 8.8: *LIG Reported Communication*

	DCS1	DCS2	DCS3	DCS4	DCS5	DCS6	DCSM
DCS1		S/R	R	R	R	S/R	S/R
DCS2	S/R		S/R	S/R	S/R	S/R	S/R
DCS3	R	S/R		S/R	S/R	S/R	S/R
DCS4	S/R	S/R	R		S/R	S/R	S/R
DCS5	S	S	S	S		S	S
DCS6	S/R	S/R	S/R	S/R	R		S/R

Figure 8.9: *DCS Actual Communication*

Figures 8.9 and 8.10 show the actual configurations set up by users as obtained from log files. Due to a misunderstanding with a user, the log file for LIG6 was accidentally deleted which results in that row being left blank.

When comparing the actual results with the reported results a different story emerges. In fact the participants almost always set up receiving configurations but did not report them.

	LIG1	LIG2	LIG3	LIG4	LIG5	LIG6	LIG7	LIG8
LIG1		S/R	S/R	S/R	S/R	S/R	S/R	S/R
LIG2	S/R		S/R	S/R	S/R	R	S/R	S/R
LIG3	S/R	S/R		S/R	S/R	S/R	S/R	S/R
LIG4	R	R	R		R	R	R	R
LIG5	S/R	S/R	S/R	S/R		S/R	S/R	S/R
LIG6	-	-	-	-	-		-	-
LIG7	R	R	R	R	R	R		R
LIG8	-	S/R	S/R	S/R	R	R	R	

Figure 8.10: *LIG Actual Communication*

This effect could be caused by the fact that they were not the actor responsible for messages actually being generated for these rules and as such they did not feel responsible for reporting these configurations. The result of this finding is that user reported configurations should be carefully considered alongside any logged data available to confirm behaviour.

Guideline/Observation 33. *Users tend to under-report usage of applications that they did not initiate themselves.*

Two interesting uses of the system emerged from this analysis. LIG4 and LIG7 both set up their configurations to receive messages only but not to send them - this allowed them to monitor all messages that they were capable of receiving but they would not contribute any messages of their own. This behaviour is typical of the *lurker* [163] demographic found in online communities. Although this behaviour was not initially expected from a group of participants who already knew each other it is not unsurprising as this application is essentially a small online community.

In contrast to this was a sending only approach, found with DCS5, wherein the user was happy to send messages updating people of their status but were not interested in receiving other peoples messages. This is similar to some behaviour on recent social networking sites, such as Twitter [115], where users can post comments without necessarily receiving other peoples comments.

Guideline/Observation 34. *There are wide varieties of different types of messaging behaviours that can be observed in the wild. Two prominent behaviours identified here are the 'Lurkers' and 'Posters'.*

8.4 Investigations into Users Configuration Behaviour

8.4.1 Evaluation Objectives

The second investigation explored user interaction experiences during the course of using this application and looked into user's requirements, concerns and expectations when using the demonstrator application in their own homes.

This allowed observations to be made about the best ways of presenting configuration information and choices to the user as well as identifying key issues that should be addressed during deployments of adaptive systems in general. From this observations and recommendations or guidelines to future work in this field are made.

This second longitudinal investigation was conducted with less technologically experienced, elderly, users and involved the installation of a similar application, built within the MATCH framework, into their homes over a longer period of time (8 to 12 weeks). This application is described in the next section. By deploying this application in the home, and observing and recording responses and behaviours at the start, mid point and end points of the investigation via interviews and observation, it allowed participants within the investigation more frequent interaction opportunities and allowed participants time to explore which approaches worked best for them.

8.4.2 Procedure

8.4.2.1 Participants

Participants were recruited via existing end user volunteers with the University of Glasgow Department of Computing Science. Four people (3 female, 1 male) self selected themselves as respondents to our search for participants; two each from Perth and Glasgow. All the participants and spouses were classed as elderly (ages 68-76).

None of our participants classified themselves as being advanced computer users although they were all found to be comfortable with the use of computers, technically adept and curious about new technologies. This is to be expected by their volunteering to be a part of the investigation.

In this study presented in Section 8.4 participants are identified using the notion R*x* (participant/respondent *x*) representing an anonymised participant in the investigation. The notation *I:* within quotations is used to indicate questions from the interviewer.

R1 was female and lived alone in the Glasgow area. She had low decibel hearing loss but remained involved in local community groups and charities. She suffered long-term pain from her hip for which she was receiving medication resulting in a decrease in mobility. She was scheduled for hip replacement operation two weeks after this investigation concluded. She used the application for 8 weeks in total.

R2 was female and lived with her husband in Perth. Her ex-fireman husband has hearing problems but she does not. They are both active and take frequent holidays and day trips. She used the application for 11 weeks in total.

R3 was male and lived with his wife in Perth. He was ex-military and still very active although had slight eyesight problems. He was very technically adept in general and was an amateur radio operator. He used the application for 11 weeks.

R4 was female and lived alone in the Glasgow area. She had had a knee operation in the previous year and still had difficulties using stairs. She suffered from poor eyesight but was otherwise healthy and used the application for 4 weeks.

From these descriptions it should be obvious there were no serious disabilities or mobility problems although classical signs of ageing were present in all cases; including reduced hearing, vision and movement.

Each participant nominated four friends or family that they wanted to be able to communicate with; mostly choosing contacts who were geographically distant. Examples are a daughter (R4) who lived in the same city, friends who lived in other UK cities (R1, R2, R3) and relatives in other countries (R2).

8.4.2.2 Tasks & Context of Use

Participants were given an extensive introduction to the application and each of its features. The introduction involved an installation into the participant's home, including creation of accounts for friends and family to use on the associated website. An hour-long introduction and familiarisation session was conducted - explaining each feature along with examples. A detailed help guide and manual was provided for the participants, which contained detailed walkthroughs of the application with images along with summary images of relevant screens and features as provided in Appendix D.

After a short period of one to two weeks (based on participant availability) a follow up intervention was conducted to troubleshoot any problems encountered by the participants; technical or otherwise. For one participant (R2) a second intervention was arranged due to

a technical failure: the UMPC reported a failed memory module at startup and this UMPC was replaced.

The intended deployment period was four weeks in total, although participants were permitted to keep the application for up to 12 weeks in order to fit into their schedule and availability more easily or if they wished to use the device for longer. At the end point a debriefing interview was conducted with the participant.

During the course of this investigation the participants task was to configure the application as they saw appropriate to enable communication with their friends and family. A user experience diary was included in the pack if users wished to report behaviours or incidents during their period of use.

8.4.2.3 Evaluation Platform

The application deployed here was motivated by the same considerations as the previous investigation which was designed to allow the participant to use the application to communicate with friends. The deployment of this application within a home environment requires that some changes to the application were made to fit more naturally into a home environment.

In recognition of the fact that this group of participants would not know each other in advance the architecture of the communication framework was redesigned from the unconstrained graph, shown in Figure 8.1 and reformed as a hub and spoke design as shown in Figure 8.11 using the participants own 'groups'. This redesign was motivated by the notion that one participant in the investigation could (and would) choose themselves the other stakeholders (friends/family/carers) they wished to communicate with. In this figure, participants are represented by the identifiers A and B and externally communicating stakeholders with the identifiers A1, A2 and so on.

The application was implemented as a digital photo frame style device on a UMPC, shown in Figure 8.12 and shown in detail in Figure 8.13, used by the participants to display their own photographs. This was intended to make the application as unobtrusive as possible while still allowing it to be placed within busy areas of the home. This choice was made coupled with the understanding that computer ownership within the elderly age range is substantially more limited - this made installation on personally owned computers a potential problem which led to the provision of the application on a dedicated UMPC machine.

The UMPC contains a touch sensitive screen; to interact with the device the participant taps
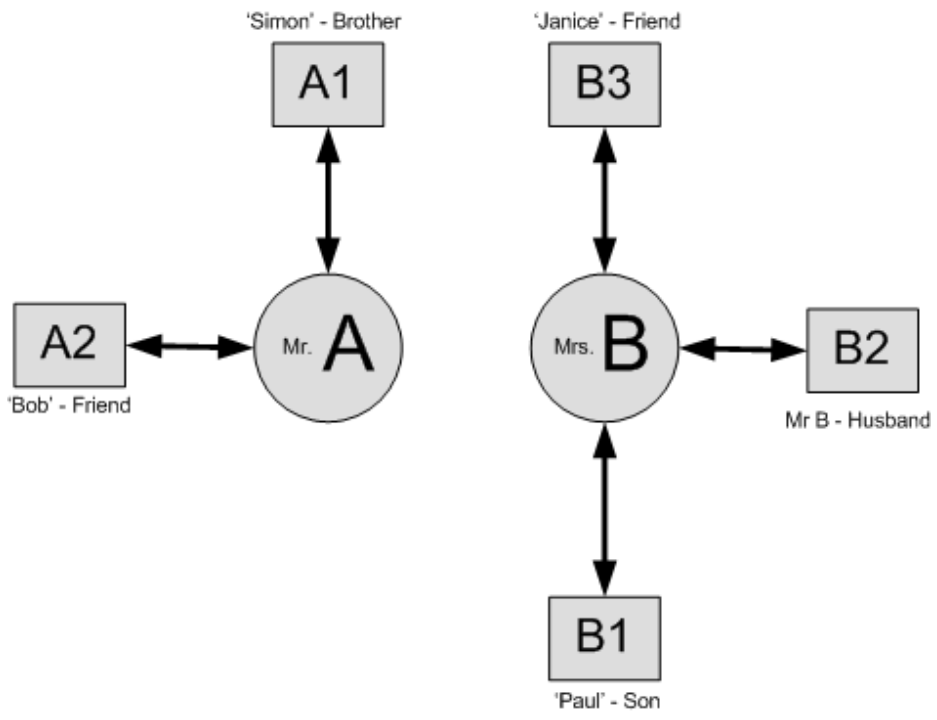
Figure 8.11: *Investigation Architecture*
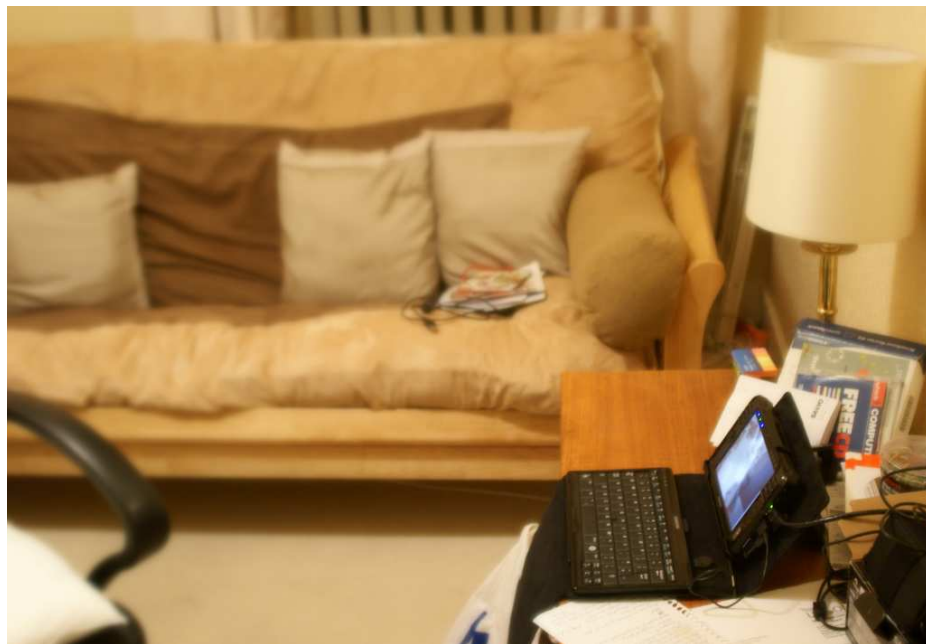
Figure 8.12: *Photo Frame Application in Context*

Figure 8.13: *Samsung Q1*

the screen once to hide the currently displaying photos and access the menu system. When this is done the screen from Figure 8.14 is displayed.
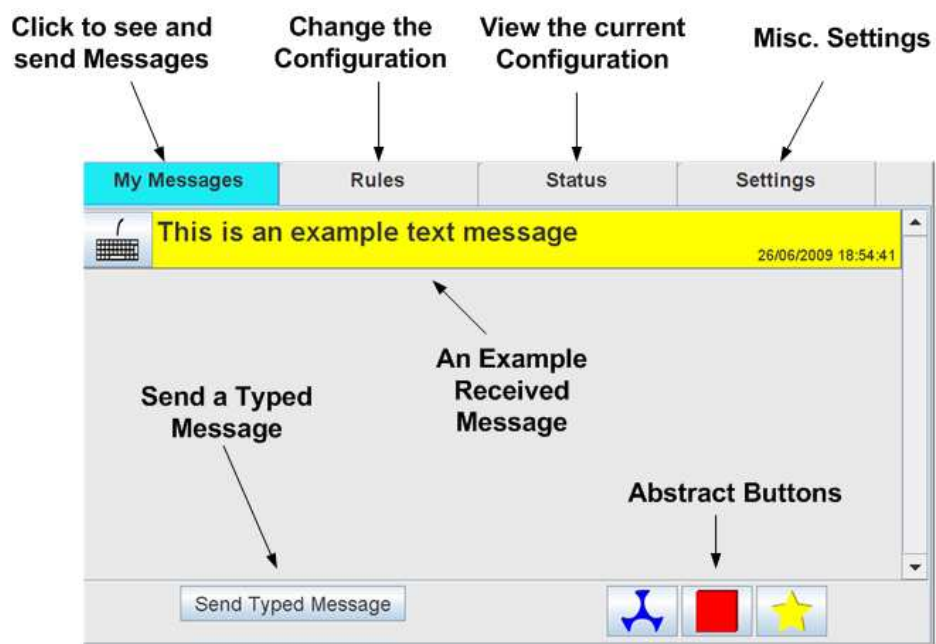
Figure 8.14: *Redesigned Main Panel*

This panel is a reworked version of the one shown previously in Figure 8.3. Font sizes were increased to make them easier to read and the layout was redesigned to use the full display area of the UMPC.

Due to the touch screen nature of the device (and the poor usability of the built-in keyboard) a touch screen keyboard was used as shown in Figure 8.15 in order to allow participants to send personal messages. The onscreen keyboard was derived from a library provided by Claire Maternaghan of the University of Stirling.

Figure 8.15: *Touchscreen keyboard*

The four tabs along the top of the screen are used to switch to other sections of the system. Figure 8.16 shows the revamped Configuration screen shown previously in Figure 8.4. The different configuration approaches were distinguished and allocated icons by function. The icons were coloured to make them more visually distinctive. The number of rules that could be set up was limited to use a fixed number (8) selected via the vertical tab buttons, due to screen space considerations, as shown in Figure 8.16.

A Status view was added which provides an overview display of input sources and output destinations currently in use, based on the result from Guideline 30 in the previous investigation. Figure 8.17 illustrates a typical status screen for input devices, showing that "text messages" is the only input source selected in any rule. Clicking on the "text messages" button opens a window showing all the outputs to which "text messages" is connected.

All other input sources and output destinations, not otherwise mentioned, are included in their previous form from the previous investigation.

Webcam movement and accelerometer movement could be activated based on different levels of sensitivity (Low, Medium and High). Capabilities to support multiple accelerom-

222
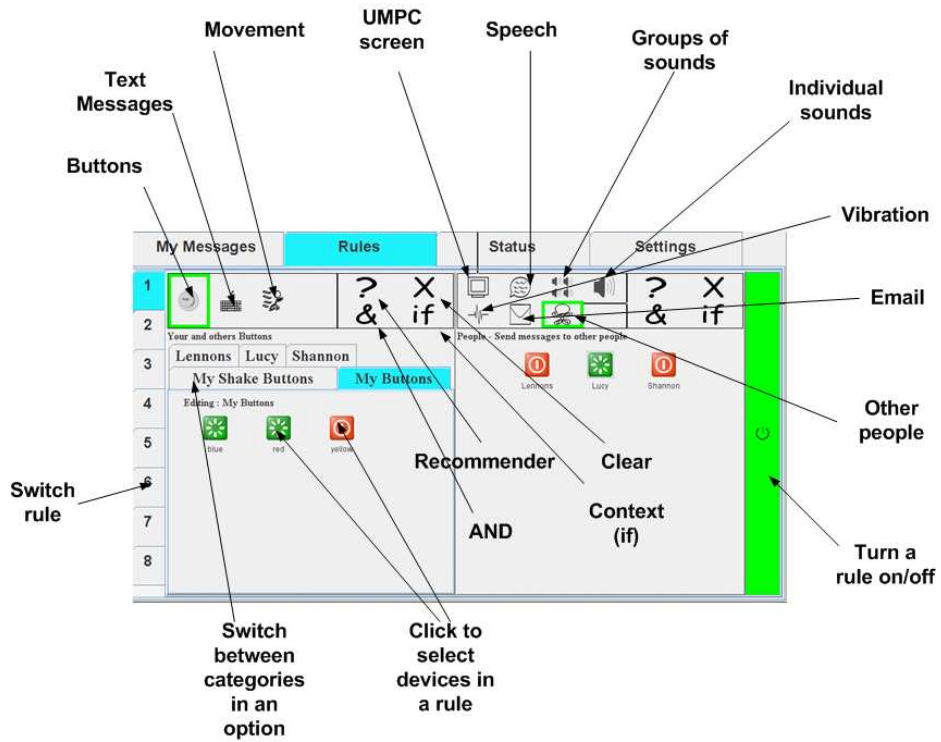
Figure 8.16: *Configuration screen*
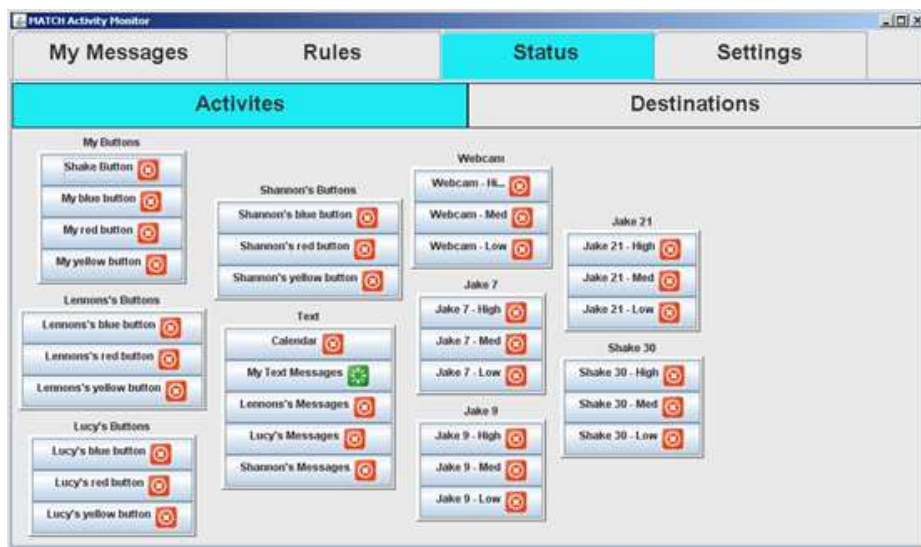
Figure 8.17: *Status screen*

eters (SHAKEs and JAKEs) concurrently were added - each with its own three levels of sensitivity.

223

The idle time at the computer input source was removed as it no longer made sense given the deployment environment (the computer running the application would almost always be idle as it was not being used for work as in the office).

Figure 8.18 shows the website developed to work alongside the application and allowed friends and family of the participant to communicate with them. External users could log onto this website to send text or button messages to the user who had the UMPC. Depending on the configuration of the UMPC these could be received using the output modalities presented in the previous investigation.
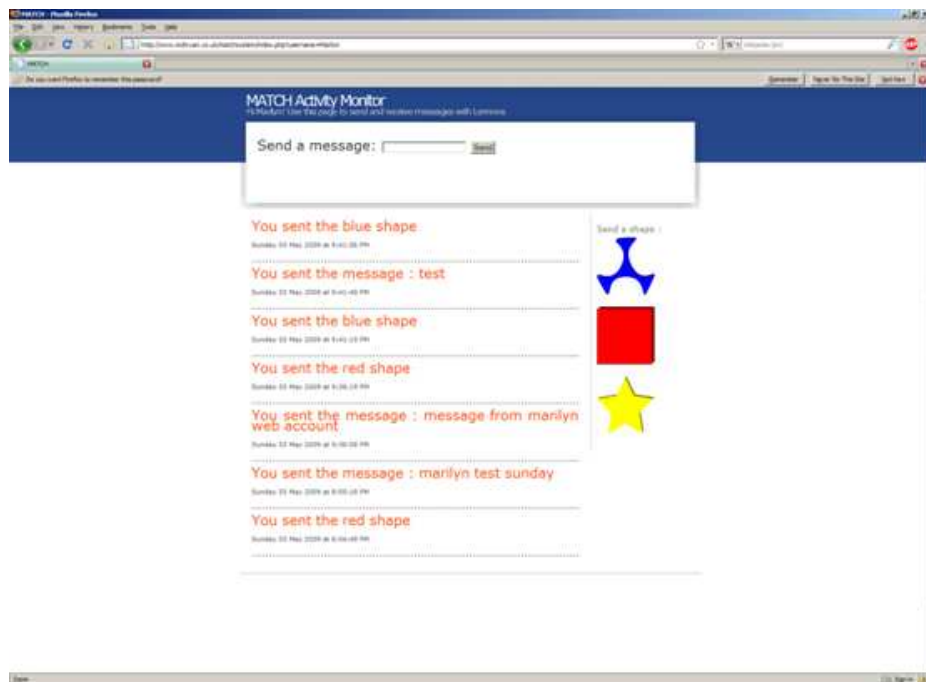
Figure 8.18: *Website*

Based on the results of Guideline 26 from the previous investigation, three *abstract buttons* were added, as shown in Figure 8.18 (the blue, red and yellow shapes on the right). These abstract buttons are presented to both UMPC and website users and allow a quick form of interaction between users. These buttons are described as abstract as no specific meaning is applied to them at development time and instead meaning is entirely decided between the recipients. This allows participants to collectively or collaboratively decide on the meaning of these buttons - allowing users to create very simple messages with predefined meanings which are similar to common concepts of "ringing the phone" to notify someone of a safe return home.

Graphical notifications of messages are provided both in a permanent list and as a visual

popup indicator over the top of photos in the photo frame mode. Participants can select these messages individually or jointly.

Auditory icons replaced the earcon reminder type and have been separated into six categories (Beeps, Birds, Gentle tones, Human, Nature and Loud). Choosing a category of Auditory icon sounds allows the application to choose one of the four sounds per category to play based on the type of message. All 24 sounds are available to be individually selected if desired, instead of using the categories.

The depth of context sensitive and combinatorial evaluation functions was limited to two levels. This was found to be the practical limit for participants in the previous investigations and allowed a simplified interface for these functions.

8.4.3 Results

This investigation was primarily interested in emerging themes from the discussion. During the framework analysis, a number of emerging themes were identified.

- Factors affecting Configuration
- Experience with Modalities
- Context Sensitivity
- Usage of Activity Monitor Application
- Learning Processes

This application operated as a valuable technological probe stimulating a rich source of relevant qualitative data. Each of the themes above is discussed in the remainder of this section.

8.4.3.1 Factors affecting Configuration

As discussed in the introduction to this chapter - the purpose of this study is primarily to look at the interaction and configuration behaviours of the participants in a home setting - particularly with respect to user interface criteria for allowing them to make configurations.

One of the major challenges concerned the design of the configuration interface. The interface that was presented to the participants consisted of a panel split into source and destination as shown in Figure 8.16.

R1: "I thought it was a bit convoluted and I felt it needed to be a bit clearer."

R3: "Basically just the process of trying to combine the two together, it's not very clear as to how they interact within the two screens and I tried to get that to sort out."

Guideline/Observation 35. *Providing several configuration screens (input and output) at once can be confusing to users.*

Participants suggested that the process that would more accurately fit with their mental model of configuration would be based on a two-stage workflow - where rules involving multiple evaluation functions (such as rules in this application which each had two evaluation functions) would be performed by selecting and configuring one evaluation function at a time.

R1: "I think it would be better to get the person first and then click on the calendar."

R1: "I think I would prefer the person that it's going to. Then decide what I was going to send to them."

This conclusion was confirmed when participants were questioned on the status screen for obtaining feedback on active configurations. This screen does use the two-stage approach they requested and users were much more comfortable using this.

R1: "I think that's better." [Referring to comparison between configuration screen and status screen]

R2: "Yes the other one was the rules. On the rules it does show you that as well but I think I tended to use the status menu."

R4: "Well, the status panel I find easier. [...] I thought it was very clear, you know, I liked that."

Guideline/Observation 36. *Split/modal status and overview configuration screens were found to be easier to understand.*

As the purpose of the status screen was to provide an alternate view onto the configuration it did not allow direct modification of the configurations but this clear preference for the status screen approach is evidence that a modal approach to configuration of more complex rules is preferred rather than a single integrated panel.

Guideline/Observation 37. *Modal approaches to configuration were preferred over persistent approaches.*

This is an interesting result as it contradicts some of the visual programming approaches in

the literature - such as Jigsaw [109] - where the entire system configuration is presented to the user at one time. It may be that this behaviour changes over longer time periods than have been examined here. Modal interfaces tend to be more structured (i.e. wizards) when compared with persistent displays (i.e. free form filling) - user preference may change over time once they become more familiar with configuration tasks.

One of the participants commented that the configuration screen was excessively graphical based in terms of icons. This was not explained as a deficit in usability but instead a feeling that iconography was more childish than the more adult alternative of more text based approaches. This serves as a general warning that efforts to make tools too friendly may lead them to feel less serious - this could be an important consideration for configuration of devices dealing with medical data etc.

> R4: "Again, you know, the overview [status] screen seems that wee [little] bit more grown up than just touching buttons."

Guideline/Observation 38. *Simple textual labels offer the user a greater sense of maturity compared to icons.*

One issue that was raised during the discussion of evaluation function configuration was the issue of security and ability to modify the evaluation functions.

> R4: "Also, building in some kind of password or something else so that folk can't just muck the whole thing up."

Although most devices in a person's home or office, which can be configured (such as thermostat, power sockets and television settings), default to a permitted state it is clear from users concerns that some access control is necessary for this type of application.

Guideline/Observation 39. *Access control of applications which affect multiple users is required.*

It makes sense that if this application should deal with potentially sensitive data then access to the application would need to be restricted. In this investigation the website portions of the application were secured by password to prevent unauthorised access to messages - however, the UMPC portion of the application presumed physical access to the device implied permission. It is clear from R4's comment above that the potential cause for concern is not actually unauthorised or malicious access but rather accidental reconfiguration that was not in keeping with her desires - i.e. "mucking the whole thing up" - through ignorance rather than malice.

Guideline/Observation 40. *Users seem to be more concerned with security as an*

227

impediment to mistakes by other potentially less informed users than against malicious behaviour.

8.4.3.2 Experience with Modalities

In addition to the themes identified with the process of actually configuring an evaluation function, this investigation was intended to explore the user preferences and subjective performance relating to the particular modalities used for interaction and identify any particular weaknesses. The three modalities which were most discussed by the participants were Visual, Audio and Movement detection.

Text/Visual

As previously mentioned there was a strong preference among the participants for textual interaction. This preference manifested itself as a higher usage of GUI based interaction and the sending of textual messages.

> R3: "Just a straightforward text message."
> R4: "I prefer text yes, when I can read it."

This preference existed even when the font size or display properties in particular lighting scenarios on the device proved difficult for comfortable reading - as was the case with any users who were partially sighted.

> R3: "They were small. I had trouble reading it. Especially in artificial light, that's not your fault this is I think my own eyesight is beginning to go slightly in artificial light."

The persistence of this behaviour, even in the face of potential difficulties, implies that this is a strongly held preference.

Guideline/Observation 41. *Despite physical impediments, such as poor vision, textual interaction is still strongly preferred over other modalities.*

When interacting with the visual parts of the system - opinion was split as to the usefulness of the stylus that was provided. One of the participants had previous experience using stylus based touch screens and did not like to use them - instead opting for a finger based touch interaction.

> R1: "I've used that thing before I don't particularly like using it..."

However, this may be largely a learned behaviour due to the quality of the stylus as it was found that some of the participants had used low quality styluses in the past which would have normally led them to avoid their use.

> R4: "I used the stylus. I liked that. I'm never confident putting my greasy fingers [on the screen]."
>
> R4: "From the sort of sat-nav, bit yeah, much prefer that. Tell TomTom [Satellite navigation manufacturer] your stylus is better."

Guideline/Observation 42. *Stylus use predominated over non-stylus use despite a recent trend in devices towards capacitive based touch screens, which do not work with styluses.*

The visual display mechanism is clearly very important within the home environment and even though some of the participants had minor visual impairments they still generally preferred using visual displays over the alternatives. This may be because of the dominant role that vision plays through an average person's life.

Another form of visual interaction provided to participants was the *Abstract Buttons* concept described in Section 8.4.2.3. The expectation that this would be used for short prearranged messages was demonstrated as being correct however user preference still sided with more descriptive textual messages. Abstract buttons were typically described without enthusiasm - this was possibly due to the need to prearrange the meaning of these messages limited the usefulness.

> R4: "I just push the button as it were and she would get the message, yes she's alive or she's got to the back door and managed to open the fridge or she's still in bed."
>
> R2: "I don't know really to be honest with you, I thought they were alright. Maybe through time you would think of others things you would associate with that person."

This conflicts with previous studies upon abstract labelling and notification conducted by [187] which showed a rapid formation of an abstract language which was used within the application. However, there were considerable differences between this investigation and the MarkerClock investigation. Specifically, participants in MarkerClock were self-selected by interest in the MarkerClock and abstract coding metaphors, were located geographically close to each other (10 minutes travel), had frequent real life communication outside of the application in order to more easily create coding languages and only communicated with one other person/household [186] (pg125). In contrast the participants here were selected based on participation in previous unrelated studies, were typically not geographically close to the people they communicated with using the application, and had a larger number of

contacts with which to communicate and may therefore have found it more difficult to establish a coding language,.

Guideline/Observation 43. *Abstract coding languages may rely on one or more of the following requirements: (i) frequent communication outside of the application to create coding languages, (ii) geographical proximity, (iii) a one-to-one relationship with the application.*

In terms of choice of images for abstract buttons the participants expressed no particular personal preferences for the type, style or themes of the abstract buttons.

> R4: "I mean, they're fine. Although, obviously you know, you could do the cute thing and put in your own personalised. You know, I'm sure my sister in law would put a picture of the dog or something, I'm quite happy with, you know the kindergarten class ... whatever symbol is given to me."

Guideline/Observation 44. *There was little preference exhibited for ability to customise the abstract buttons deployed.*

Audio

This investigation provided two categories of audio feedback to the participants. As this device used an internal speaker system, the volume of the audio output was adjusted to maximum in order to make the audio as clear and as loud as possible. However, this was still regarded as being slightly quiet with some participants who had hearing difficulties.

> R2: "I would have said like I could hear them - but my husband couldn't hear them because he's got a bit more problems with his hearing you know."

Each participant was asked for their views on the speech synthesis system used (Cerevoice with the female, Scottish accented, 'Heather' voice). One participant suggested using different voices for different types of messages.

> R3: "One, if you hear that voice you know, if you hear the voice it's a warning, the other voice it's not a warning."

Participants hypothesised that they preferred longer messages formatted to be said more politely rather than authoritative sounding messages, excepting the extreme case of messages deemed to be of very high importance where short and concise messages were preferred.

> R3: "It depends what message. Because if you've got a warning message, short and very very precise."

Guideline/Observation 45. *It may be beneficial to be able to configure multiple voices for different types or sources of messages.*

However, this raises a design issue as it was observed that very short messages are more difficult for people to correctly interpret what was said. Longer messages tended to provide the participant with a cue that they should start listening as well as more contextual clues as to the context of the message which a short and concise message lacks as the important information is delivered immediately and without warning. One possible approach to this is to combine short messages with audio cues to indicate when a message is about to be played.

> R2: "It just seemed clearer on the second one than the first one you know."
> [Referring to long and short messages respectively]

Guideline/Observation 46. *Short messages, although seemingly better suited configured to urgent messages, are difficult to understand if delivered without warning.*

Auditory icons were provided in the system as a form of audio cue. A number (24) of these auditory icons were provided in six categories (Beeps, Birds, Gentle tones, Human, Nature and Loud). Participants in this investigation strongly preferred more abstract auditory icons, such as beeps, rather than those which could be directly mapped onto particular events, such as animal or human sounds.

> R1: "I think it would need to be a ring or, in my opinion, I think it would
> be a ring or a buzz rather than a baby laughing or something."

Guideline/Observation 47. *Abstract auditory icons were preferred over concrete and recognisable sounds.*

This preference for more abstract auditory icon sounds is similar behaviour to the preference for less concrete abstract buttons; participants wanted to be able to tie any action or notification sound to a warning without the limitation of a sound being only suitable for a limited subset of notifications.

Movement Monitoring

Participants were provided with two main approaches for automatic detection of activities. For direct interaction, two different types of accelerometer based movement detection were used in the form of JAKE and SHAKE devices (as described previously).

The JAKE has much smaller physical dimensions than the SHAKE device but it was found that several participants actually preferred the larger form factor of the SHAKE due to fears that the small size of the JAKE might cause it to be lost.

R4: "Because it's so tiny - actually I was afraid of losing them."

Guideline/Observation 48. *Miniaturisation may actually hamper a technology's acceptance due to fears it could be misplaced.*

A limitation on the usage of these devices was observed due to their low battery life. Each of the devices had a battery life which ranged from 5 to 8 hours which was insufficient for a full days usage.

R2: "Yes but the thing that drove me mad about those was that they didn't really keep too long, the charge if you know what I mean you felt you were always charging them up."

This meant that users were not inclined to use them, as they could not rely on them to be functional for the entire day.

R2: "If the charge went that would give you the suggestion that they weren't moving about you know."

Another complicating factor was found as one of the participants (R2) was a licensed amateur radio operator; although the JAKE and SHAKE used different frequency allocations from the amateur radio, it was observed that their range was drastically decreased when that equipment was being used. Amateur radio power output ranges from 10-400W while the Bluetooth receiver used in the SHAKE and JAKE devices is limited to under 100mW.

Guideline/Observation 49. *It can be difficult to distinguish between a "failed device" and the "no activity" states. Even if the battery state is known to the receiver of the message they cannot use this information to assume that activity has actually occurred and not been measured. This can restrict trust in a device which senses activity.*

For this technology to be accepted in the home it must be supplied with a significantly longer available battery life or be integrated into devices which, themselves, have much higher battery capacities. A possible alternate device, not widely available at the time of deployment, would be to use accelerometers in mobile telephones - in addition to physical proximity to the user they are more likely to be kept charged if they provide other uses.

In terms of usage of the accelerometer devices it was found that some participants would attach the device to an object of interest that whey wished to be monitored or alternatively would interact with the device directly to signal a to their friends and family that a particular event had occurred.

R2: "I actually attached one to the phone."

R4: "I thought to try it out, you know and try to set it up so that I would, you know, shake as I went out the back door and she would know I wasn't at home so she wouldn't bother coming down the road."

Guideline/Observation 50. *Accelerometers were used as ad hoc movement or notification devices to detect discrete events that can be detected automatically by some movement or by some deliberate user behaviour.*

In addition to the accelerometer-based devices the system had a webcam which was capable of detecting movement within an area directly in front of the UMPC device. Participants did not express concern for monitoring of their own movement but did acknowledge possible privacy issues resulting from an indiscriminate movement detection system that may possibly affect visitors to the home.

R2: "Well I think the webcam's pretty good actually. I would say that if people were having it - if their family was away they would really like a webcam where they could actually speak and see their family and they could see them. But I could understand if it was just in the sense of the social people looking after an older person they would find that possibly intrusive, if you know what I mean?"

Guideline/Observation 51. *Users seem more interested in other people's privacy than their own - favouring function over privacy for themselves but cautioning against this being true in the general case.*

8.4.3.3 Context Sensitivity

Context sensitivity was regarded as an important feature by participants in the investigation for switching between available modalities - however, the context the participants were interested in was typically the type or severity level of the message rather than the state of the environment or of the user.

R1: "I think it would be if there was something, a warning it would have to be the louder noises and the quieter noises could be."

R1: "I would think looking at it from a point of instructions being given about medication or something then I think it would have to be short and to the point and obviously I realise if it was coming from a member of the family then it would be quite good to have that kind of message I've just heard which is quite pleasant and easy to listen to."

Most of the rules set up in the system were to direct source of messages to a different outputs based on their source rather than to direct it to an output based on the context of the user at that time. In this investigation context sensitivity selection was featured, based on movement detected recently via the accelerometers or via the webcam, but this was largely not used.

Guideline/Observation 52. *Context sensitivity is used within configurations primarily as a means to distinguish between destinations for messages rather than to select their source.*

8.4.3.4 Usage of Activity Monitor Application

As in the previous investigation, it was important to look at the usage of the system and report some of the more significant differences found between this investigation and the previous investigation.

In the previous investigation the system was deployed as a regular Windows application on an office machine already being used by the participants. In this investigation the application was a separate application on a dedicated device. This change, and the placement of the application in the home instead of the office, may be responsible for an observed behaviour of turning the application off out of habit rather than out of necessity.

> R2: "I always find I think, where I came across problems was I've got my family so trained to switch things off. My daughter was switching off before she switched off the machine."

An element of this can be attributed to participants wishing only devices which truly needed to be remained powered on overnight.

> R3: "No I switch them off at night, but during the day. I can leave them switched on if I want to, it depends. If there's a reason to be switched on, yes they'd be on all the time."

Guideline/Observation 53. *During longitudinal investigations, any applications should be designed with the notion that they will be frequently turned off at night.*

This implies some other requirements on the part of the designer; should an investigation require that an application is permanently powered on then it and its supporting applications should be designed to facilitate this. For example, it was found in one case that the application was kept on overnight, however the ADSL router that the application depended on was not. Recommendations to accommodate this could include battery-backed

applications which can operate on internal power overnight or automatic timers (via BIOS configuration for example) designed to wake the application at a specific time.

Participants already had pre-existing methods of communication with members of their friends and family which they have used in order to stay in touch. It was reported that the participants in this investigation generally did not use mobile telephones in order to communicate with friends and family.

> R2: "We have to phone now and it's really a nuisance you know."
>
> R3: "I've got a mobile phone but I'll use it as an emergency I don't, it's not switched, it is switched on, but it's not there all the time. I don't carry it in my pocket all the time - if I leave it switched on - it's switched on by mistake."

Despite this, the participants reported seeing an increasing need for a system which allows them to make low effort notifications to reassure their family members that there were no problems. A possible reason for this is that telephones are a high impact in terms of interruption whereas what they desired was actually something that involved much less conscious effort for both sides.

> R4: "It saves her phoning every few hours to check - "are you alright?", you know, "are you moving around?", "do you need anything?" - and I was able to alert her, or had been able to alert her - that would have been quite useful - rather than having to open up the computer."
>
> R4: "You know - she got the red button - she says "that's fine, mothers managed the stairs", or on the other hand she gets nothing at all."

Guideline/Observation 54. *Despite existing communication mechanisms there are still increasing needs for status, and in particular well-being, awareness applications which are low effort.*

Participants indicated that they were comfortable about receiving notifications for a large number of events even though they would be interested in only some of them at any one time but which they would mentally filter. One participant related it to listening to many different voices in a room or radio and mentally selecting the voice of interest and discarding the rest.

> R3: "Suddenly you hear that one voice, or that one thing, [and] you know switch on. So that kind of system, whereby you've got two voices but one is a, grabs your attention, the other one doesn't."

This approach of using the system matches a low interaction requirement where the participants wanted to use the system to automatically notify their contacts about their condition with little or no interaction required on their own behalf. It is this style

of interaction that they have identified that they see a use for and which their current approaches (mobile phones, email) can't deliver as they require concious effort to notify other people of their condition. This should be further investigated in related home care studies.

8.4.3.5 Learning Processes

Due to the large number of features deployed in this application, potential problems with a steep learning curve had been foreseen and extensive steps to ameliorate this were taken. During the initial installation an hour long introduction and familiarisation session was provided to explain each of the features, along with a help guide and manual with detailed image based walkthroughs of configuration as well as summaries of relevant screens within the application and the available features. In addition, the self-selected group were technically competent and eager. It was believed these factors would be sufficient; however despite these efforts it was still found that participants experienced a steep learning curve.

> R1: "I felt I would have liked a bit more instruction, maybe even that's basic, but you know I felt if I'd had that that would have helped."
> R2: "I didn't have time to sit and understand it."

One possible explanation for this is that the system deployed included a large range of interaction techniques - both sources and destinations for activities. It is possible that the sheer range of new technologies introduced all at once was sufficient to intimidate the participants. The reason the same effect was not observed in the previous investigation may be because those users were already familiar with the concepts for a number of the devices used and did not need to make new mental classifications as to the capabilities, features and drawbacks of a particular interaction technique.

> R3: "I discovered a long time ago, you get the problem up and running, you get it foolproof and then add the bits and pieces on."

Given this, investigations that deploy a large number of novel technologies should be introduced in phases even if the new technologies are not necessarily the focus of the investigation in order to prevent unfamiliarity with the devices from presenting a steep learning curve.

Guideline/Observation 55. *Technology investigations involving large numbers of new or unfamiliar technologies should be introduced in phases.*

8.5 Overview

This chapter detailed the results of studies involving an application which was deployed containing the model described and tested in previous chapters. This study was designed as an exploratory probe to see what behaviours and actions people adopted when they encountered this technology in their home.

An empirical study of the configuration behaviour of the participants in the field and over time reveals that behaviour closely resembles the evolutionary model proposed in this thesis. Behaviours have been identified consistent with each of the four key stages of Interaction Evolution. Configurable and adaptive systems should therefore aim to support Interaction Evolution and encourage or promote end user configuration over time.

The key aim of the second study was to identify preferred approaches for presenting configuration information to users. The segmented approach of selecting inputs or outputs in stages that was adopted in the Status/Overview screen was found to be preferred over the single screen approach utilised in the configuration screen.

In addition, a large number of guidelines and observations have been presented to help guide future work in the field and help other practitioners to make informed choices when considering the deployment of longitudinal studies of this nature.

Specifically, the following important observations were made about participant behaviour during configuration as a result of the first investigation:

1. Behaviour consistent with the four stages in the process identified in Chapter 4 (Identification of opportunities, Reflection on alternatives, Decision Making and Implementation) were identified.

2. The model described in Chapter 5 was shown to support these four stages by allowing them to be expressed in a running application.

3. Not all configuration choices can be treated equally - a high degree of control is important, especially where selection is guided by non-functional attributes, but ease of use is more valued for selection between homogeneous entities.

4. User's feelings of and perceptions of control are correlated with the ability to directly observe the behaviour of a configuration. Configuration techniques which explicitly showed what they were currently doing and had immediate effects resulted in higher perceived levels of control. This demonstrates the need for the ability to directly inspect the current configuration in addition to the rules that have been set.

5. There is a wide variety of preferences between users in terms of : trade offs of control vs. ease of use, viewpoints towards recommenders, context sensitivity, and mental models in terms of the most important foci for configuration (inputs vs. outputs, people vs. devices). This reinforces the need for a variety of different types of configuration approach for different people (implemented as evaluation functions).

6. Configuration occurs as a combined result of evolutionary changes to test new configurations and immediate changes inspired as a result of realisation of an incorrect configuration and knowledge of how to fix it.

7. Some devices excel at particular purposes (audio for alerts) and their uses tend to be restricted to these purposes.

8. There was a significant amount of discussion between participants on what the application could be used for, how it could be configured and sharing of configurations. Coding schemes were created between pairs of participants.

Similarly, the second follow up investigation provided a number of observations about the experiences, preferred methods and techniques and factors affecting the success of a configuration.

1. There is a high dependency on visual media, even when participants may be visually impaired.

2. Workflow approaches involving a configuration screen which is split into multiple stages rather than presented all at once were preferred. An example of this was the overview screen which was split into multiple stages and was preferred by participants.

3. Access control and security were considered as means to prevent accidental misconfiguration rather than malicious activity,

4. Abstract coding behaviours were found to rely on one or more specific requirement (such as close physical proximity or high frequency of communication) in order to be formed between pairs of participants.

5. Abstract auditory and visual icons were preferred over more concrete forms for both audio alerts as well as messages to send to other participants. Accelerometers were used to instrument arbitrary discrete events. Taken together this implies a preference for general purpose devices and interaction methods which can be appropriated for user specific functions rather than sounds, icons or sensors designed for a specific function.

6. These abstract notifications were desired as a low-cost background notification service which would require little effort. Existing technologies (telephones) already satisfy requirements for long involved communication.

7. Learned behaviours from similar devices can influence preferences of new or novel devices - either encouraging (Twitter) or discouraging (Touch screen stylus) its usage.

8. Some types of message (long speech samples) may only be effective when combined with other modalities (chimes to indicate upcoming message) which shows that configurations need to be able to be combined.

9. Context sensitivity is used primarily for controlling output destinations - input sources were considered as being part of the purpose of the task rather than something that would change during the task life.

These findings substantiate the claim that the process presented in this thesis is a true and accurate depiction of configuration processes employed by users and that both the model and implementation support this process. A large number of supplemental findings were obtained which support the assertions made previously in this thesis that users desire the ability to personalise configurations to meet their specific need and are happy to appropriate applications to configure and customise them for their own purposes and that this configuration occurs as an evolutionary process.

9

Future Directions

The work presented in this thesis has revealed a number of potential directions for future work. Specifically, five areas of research are particularly interesting candidates; (i) generalisation of the approach to evaluation to domains outside interactive systems, (ii) improvements to graph traversal algorithms to improve performance of the approach, (iii) additional formal verification and analysis, (iv) integration of formal modelling into the model and (v) application of the model to challenging system contexts and tasks.

This chapter will briefly discuss each of these directions.

9.1   Generalisation

This thesis concentrates on the area of interactive systems, and specifically Ubicomp applications, in relation to the process and model presented here. However, it may be feasible to apply the model to more general problems of choosing an item (or items) from a set of possible options, particularly one that may change frequently.

In order to approach this generalisation, it would be necessary to reformulate the model presented here as a more general pattern which is not restricted to interactive systems. Broadly speaking, the approach could concentrate on formulating equivalent approaches to

determining the available sets of possibilities that can be chosen.

The model should be amenable to supporting choices from domains which exhibit similar behavioural characteristics to interactive systems; viz. (i) there is a large but not unbounded set of options, (ii) it is possible to derive these options mechanistically, (iii) there are a large selection of methods and techniques can be devised to filter, rank and analyse the available options, and, (iv) optionally, these methods can be combined together to increase the power of the methods.

Three example domains are described here;

- **Custom item store front** - There exist many products which can be custom built for an individual customer. One example is custom computer building, as practised by Dell, where the customer is capable of making modifications to an initial system configuration. Different components (video cards, hard disks etc) may only work given a particular choice of other component (motherboard etc). These dependencies can be modelled in graph form and traversed to determine all possible configurations. These configurations can then be filtered and ranked by different criteria (price, speed, availability, future expansion ability, reviews, popularity with other customers). These criteria can be combined to facilitate a customer to find the computer of their wishes.

- **Recipe selection** - An application could be built with a list of recipes represented as a graph to indicate their requirements (ingredients) in graph form. The graph could be traversed to determine the set of possible recipes given the available ingredients in the kitchen. These recipes could then be ranked by a series of criteria (taste, effort, time to cook, cost of ingredients, type of meal, time of day, current temperature) which are ultimately combined to decide which recipe should be selected for the meal.

- **Game move computer** - Chess computers and similar systems work by exploring a tree of possible moves and selecting the ones most likely to lead to victory. This tree could again be represented as a graph with each move as a node within the graph. Criteria in chess broadly break down into position and material and include criteria such as checking for a checked king within the suggested move, pawn advancement, proximity of pieces, number of other pieces under threat etc. These criteria need to be combined in order to choose an appropriate move.

The example domains selected here are not necessarily practical but, rather, are for illustration purposes only. They demonstrate that the general approach has merit in terms of the possibility of applying it to other application domains which may have very different criteria for selection.

9.2 Integration and Performance

One algorithmic challenge is the efficient determination of available possibilities - derived from a graph traversal operation on a graph representing the available resources. In the model presented here, the model uses a depth-first traversal search of the graph. This is essentially, the simplest and easiest to understand traversal that can be applied but is far from the optimal approach in terms of performance.

Faster and more intelligent approaches to this problem are available such as the A* algorithm [104, 105] which is a type of informed search algorithm. A* (and similar algorithms) rely upon a heuristic estimate of the quality (usually distance to the goal) in order to determine which nodes to visit in the graph. Although it is not as powerful (as it can only examine individual elements within the graph rather than an entire possibility), the heuristic estimate could be replaced by an evaluation function (or similar construct) which is able to intelligently direct the traversal of the graph and, optionally, perform filtering of undesired possibilities during the traversal. This approach may have significant performance improvements as a result of integrating an additional evaluation function phase into the traversal of the graph.

Another approach is to utilise specialised computational models for efficient graph processing such as Google Pregel [136] by Malewicz et al. Pregel is a computational model which expresses programs as a sequence of iterations where messages can be passed between iterations and the graph can be modified or mutated at each stage. Pregel is capable of performing common graph operations (such as depth-first search used in this model) upon graphs with billions of nodes and trillions of edges by allowing efficient division of work between commodity multi-core PCs. One problem discussed by Malewicz et al. is the calculation of a single source shortest path, a similar problem in principle to the requirement of our graph traversal, which was executed on a randomly generated graph of 1 billion nodes and 80 billion edges on a cluster of 480 PCs within less than 200 seconds. This shows that efficient computation of graphs with many orders of magnitude more nodes and edges than would be expected within a ubiquitous system can still be considered tractable.

9.3 Verification

9.3.1 Additional modelling

As discussed in Section 7.4.3.8, the implementation of the model presented in this thesis was used within the *Verifying Interoperability Requirements in Pervasive Systems* (VPS) project which identified three example approaches where formal modelling can be used to reason about configuration choices. These were;

- **Conflicting rule detection** - Formal modelling could attempt to detect conflicting or nonsensicle configurations which have been configured. This could include situations where rules or evaluation functions have been configured in mutually exclusive ways, where an unshareable resource is required by multiple entities, or where usability requirements are degraded given the current configuration.

- **Redundant rule detection** - Evaluation function configurations may have some overlapping or repeated definitions between tasks and it is advantageous if such redundant configurations can be detected to provide feedback to the user or to remove them from the active configurations

- **Modalities** - Input and output components can be classified by modalities and the acceptability of such a system may depend on correct use of different modalities. It is possible to validate that evaluation functions are correctly choosing only appropriate modalities for the user in question; i.e. visual output devices should be avoided for severely visually impaired users.

- **Priorities** - Some messages or interactions may be deemed of higher importance than others and it can be useful to check that there is no overlap between forms of high and low priority messages.

Currently, the only approach that has been taken by the VPS project is to implement redundant rule checking of the implementation using Promela/SPIN models and SAT solver applications. One obvious avenue for future research is to address the remaining two applications and identify other applications where formal modelling of configuration choices can perform useful reasoning. Other possible applications could be sought from areas of functional behaviour, security and performance etc.

9.3.2 Integration of formal modelling

The goal of the VPS project is tightly coupled verification of configurable systems and configurable models. This process involves a feedback loop between the user performing a configuration and formal verification of the configuration.

The VPS project imagines that a user configures the application which is then comprehensively analysed using the techniques detailed in the previous section. The analysis may then suggest improvements or other changes as a result of the analysis and this information is then used to reconfigure the system again; either automatically by prompting the user to inform them of the results.

One avenue of research is to integrate the formal modelling approaches into the process and model described in this thesis. One possible approach could be to create evaluation functions which implement the formal modelling techniques as evaluation functions. These could then be used to evaluate, and change, the state of the application configuration and would allow feedback to the configuration directly from the formal analysis functions.

This would provide for extremely tight integration of formal modelling into an application.

9.4 Application of the model

In addition to the further work highlighted above which concentrates on improvements to the model itself, there are also a number of application-oriented features which could be addressed in further work. These include investigating the best ways of presenting, implementing and allowing users to use novel features that are made available by the underlying model.

Amongst these, particularly suitable aspects of research would include:

- **Deferment** - The model explicitly supports evaluation functions which can defer their choice of possibilities until a later point in time. The evaluation function can then signal when it has gathered sufficient information to make a choice. How is this best presented to users and what would they use it for? What should be done in the meantime? How should deferred evaluations be presented to users so that they can incorporate them into an evaluation function composition?

- **Collaborative Evaluation** - The model is capable of combining together a variety of criteria but these criteria could also be sourced from multiple different users. How should the combination of potentially conflicting criteria take place where there may

not be a single person "in charge" to mediate the combination? Which types of evaluation function should be used to combine these criteria? As in Section 6.5.4 there is no general solution to this problem but the question remains of how the different approaches and drawbacks can be presented to the user.

- **Discovering the most useful evaluations** - How can the most useful evaluation functions (or combinations of evaluation functions) be discovered? The studies in Chapter 8 represent initial work on determining which styles of evaluation function people prefer to use however there is still an open question of which evaluation functions are actually the ones that people want to use on an ongoing basis and why? Further research would assist in determining what the relevant factors and criteria are and in which contexts or circumstances they apply.

9.5   Overview

This chapter has suggested a number of future directions which emanate from the work presented in this thesis but there are many more that could be investigated. Each of these are interesting research areas; particularly in the area of investigating optimal presentation and usage of the features made available by the model. The ongoing VPS project (discussed previously) is conducting active research in the area of integration of formal modelling with model based systems and is grounding some of its work on the work presented in this thesis.

10
Conclusions

There have been a number of contributions made in this thesis. The major contributions of this work are:

- **Chapter 3** - A study into the requirement for systems which can support evolutionary configuration.

- **Chapter 4** - A discussion on the process of evolutionary configuration.

- **Chapter 5** - A model designed to support this process.

- **Chapter 6** - An extension to the work of Thevenin and Coutaz resulting in a taxonomy and characterisation of the configuration space.

- **Chapter 7** - An exemplar implementation, incorporating techniques from the characterisation, and demonstrating the feasibility, validating the ideas in this thesis and which served as a development platform for a variety of other projects.

- **Chapter 8** - Two longitudinal investigations exploring the processes and methods employed by users when engaged in configuration tasks and discovering the relevant factors which affect the success of a configuration and subsequent configurations.

- **Chapter 9** - Identification of future directions that this work can be taken. Some of which are under investigation by the VPS project.

This dissertation started with a thorough examination of the literature on change in adaptive systems in Chapter 2 and, in conjunction with an experiment performed in the context of audio reminders in Chapter 3, was able to identify a need for systems that can evolve as well as identifying a lack of systems which can currently accommodate this need satisfactorily.

A process of *Interactive Evolution* was identified in Chapter 4 which is consistent with the definition explored in Section 2.1.4. This process involved four key components; (i) Identification of opportunities for change, (ii) Reflection and judging of alternatives, (iii) Making a decision from the set of alternatives, and (iv) implementation of the choice followed by iteration of the process.

Based on this process an Interactive Systems modelling approach was taken and a model illustrated in Chapter 5. This model introduces the concepts of *possibilities* and *evaluation functions*. This model is discussed further in Chapter 6 where a characterisation of the configuration space is presented which builds upon previous characterisations by Thevenin and Coutaz. Specifically, the model is shown to support configuration within a multidimensional configuration space defined by five axes; (i) Target, (ii) Source, (iii) Means, (iv) Time and (v) Actor.

This model was implemented using the Java OSGi platform. This implementation is believed to be a particularly efficient and flexible approach to interactive systems design and contains a number of features making it an valuable testbed for longitudinal investigations. The implementation is extensively validated and has been used in eight other projects; including two Masters theses [119, 228], one other PhD thesis [55] and in published work by the EPSRC funded *Verifying Interoperability Requirements in Pervasive Systems* (VPS) project [4, 31].

The implementation was used to build two activity monitoring applications which leveraged the features and capabilities of the implementation and were used in two longitudinal investigations to investigate the processes, methods and approaches used while configuring a large or complex system and to discover the relevant factors which affect the configuration process and the success or failure of particular configurations. These investigations uncovered a number of interesting and relevant factors affecting configuration choices, reinforcing the need for change, reiterating the importance of the process model presented earlier and raising important issues and considerations that should be addressed when designing for change. These investigations highlighted the similarity between the configuration process described in this thesis and the one actually used by users when interacting with the system and showed that this was supported by the model and implementation.

Finally, this thesis has highlighted a number of future directions within which this work could be taken. Specifically, (i) generalisation of the approach to evaluation and selection of options from outside interactive systems, (ii) improvements to graph traversal algorithms to improve performance, (iii) additional formal verification and analysis that could take place, and (iv) integration of formal modelling into the model.

This thesis set out to answer the question in Section 1.1. Specifically the question posed was:

- *How can system change be modelled and implemented in order that the system can enable a user to answer the questions:*

    - (i) What is the system currently doing?

    - (ii) What can it do?

    - (iii) How can it be changed?

These questions have been addressed within this thesis as follows:

What is the system currently doing? : The mechanism for evolution in this thesis explicitly models configuration options as *possibilities* which can be enumerated and inspected and upon which evaluation functions operate. The possibilities currently active in an implementation can be directly inspected which allows the user to determine the current configuration. This has been demonstrated to be of particular use within the second investigation which showed that direct inspection of the status of the system was important for a feeling of control.

What can it do? : A novel approach for the determination of the currently available set of configuration options (modelled as possibilities) via a graph traversal operation was presented. This allows an exhaustive set of possible alternate configurations to be determined which can be presented to the user or can be inspected, filtered, ranked and sorted by evaluation functions on behalf of, or in conjunction with, the user.

How can it be changed? : The mechanism for changing the current configuration is undertaken by control and manipulation of evaluation functions which are responsible for selecting the possibilities to be implemented within the application. Evaluation functions operate on behalf of the user (or users) and represent their requirements and preferences in for how the system behaves. This allows users to moderate and control the evolutionary process and to dictate how the system can change. Users were shown to have a variety of different mental models, during both investigations, for how configuration should take place and thus need a variety of alternate approaches to controlling configuration.

The first experimental investigation conducted supports the claim that this is a useful approach to the presentation of this information to end users by demonstrating that the mental models and experiences of the users correspond to to the mechanisms used within the configuration process and evaluation model that have been presented.

Finally, a number of specific supplementary claims were made. It was claimed that evaluation functions can support varying modes of use and can be combined to allow novel support for configuration of interactive systems. This is demonstrated in a variety of places within this dissertation; methods for varying modes of use are discussed in depth in Chapter 6 where a large number are presented and shown to interact with the model and where the implementation of the model supports such usage - demonstrated by a variety of modes of use for different configuration approaches used in the implementation and longitudinal investigations.

The characterisation of the configuration space in Chapter 6 shows that this approach is capable of representing a large variety of currently available techniques for the configuration of interactive systems. This is evidence towards the claim that this approach allows for systems that support a superset of currently available techniques for configuration of interactive systems. This claim is further substantiated by the large number of applications created within the implementation in Chapter 7 by other students and researchers.

Implementations of different configuration approaches within the implementation demonstrate the ability to provide greater flexibility in terms of configuration technique. The model and implementation are capable of dynamically changing configuration technique or configuration criteria at runtime and of supporting a wide variety of configuration techniques.

It was claimed that the approach provides users with information on the system capabilities and status which can be difficult to determine otherwise. The model has been designed with this requirement specifically in mind and represents this information explicitly within the model and is capable of presenting it to end users. One form of this ability is the Status screen in the second longitudinal investigation. This information may be difficult to determine otherwise and allows users to make more informed choices about system behaviour.

In summary, it is believed that the work conducted in this thesis answers the research question that was originally posed: a versatile system has been developed which models change and allows users to answer the three sub-questions as detailed above. Furthermore, this system offers an approach to modelling change that is flexible and can be used to encapsulate a large variety of techniques and criteria for configuration.

Bibliography

[1] Gaudenz Alder. *The JGraph Swing Component*. Diploma Thesis, Federal Institute of Technology ETH, Zurich, Switzerland, 2002.

[2] Cliff Allen. Personalization vs. Customization. *http://www.allen.com/cgi-bin/gt/tpl.h, content=26*, 2003.

[3] Apache Software Foundation. Apache Felix OSGi Release 4 (Felix 2), 2009.

[4] Myrto Arapinis, Muffy Calder, Louise Denis, Michael Fisher, Philip Gray, Savas Konur, Alice Miller, Eike Ritter, Mark Ryan, Schewe Schewe, Chris Unsworth, and Rehana Yashmin. Towards the Verification of Pervasive Systems. In *Proceedings of the Third International Workshop on Formal Methods in Interactive Systems (FMIS 2009). Electronic Communications of the EASST.*, volume 22, pages 21–31, Eindhoven, NL, 2009.

[5] David Arnold, Bill Segall, Julian Boot, Andy Bond, Melfyn Lloyd, and Simon Kaplan. Discourse with Disposable Computers: How and why you will talk to your tomatoes. In *Usenix Workshop on Embedded Systems (ES99)*, 1999.

[6] Ken Arnold, Robert Scheifler, Jim Waldo, Bryan O'Sullivan, and Ann Wollrath. *Jini Specification, 1st edition*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[7] Kenneth J. Arrow. A Difficulty in the Concept of Social Welfare. *Journal of Political Economy*, 58(4):328–346, 1950.

[8] Ernesto Arroyo, Ted Selker, and Alexandre Stouffs. Interruptions as multimodal outputs: which are the less disruptive. In *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces (ICMI)*, pages 479–482, Pittsburgh, USA, 2002.

[9] Matthew P. Aylett and Christopher J. Pidcock. The cerevoice characterful speech synthesiser sdk. *Lecture Notes in Computer Science*, 4722:413, 2007.

[10] Lars Baeckman, Brent J. Small, and Åke Wahlin. Aging and memory: Cognitive and biological perspectives. *Handbook of the psychology of aging*, 5:349–377, 2001.

[11] Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, and Gaëlle Calvary. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In *Second European Symposium on Ambient Intelligence*, pages 291–302, Eindhoven, NL, 2004.

[12] Roland Balter, Luc Bellissard, Fabienne Boyer, Michel Riveill, and Jean-Yves Vion-Dury. Architecturing and Configuring Distributed Application with Olan. In *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)* , pages 15–18, The Lake District, UK, 1998.

[13] Simon Banbury, Liz Fricker, Sébastien Tremblay, and Lucy Emery. Using auditory streaming to reduce disruption to serial memory by extraneous auditory warnings. *Journal of Experimental Psychology: Applied*, 9(1):12–22, 2003.

[14] Simon P. Banbury, William J. Macken, Sébastien Tremblay, and Dylan M. Jones. Auditory distraction and short-term memory: Phenomena and practical implications. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 43(1):12, 2001.

[15] Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. *OpenSSH, The Secure Shell: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[16] Len Bass. Metamodel for the Runtime Architecture of an Interactive System. The UIMS Tool Developers Workshop. *Special Interest Group on Computer Human Interaction (SIGCHI) Bulletin*, 24(1), 1992.

[17] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn A. Stein. OWL Web Ontology Language Reference. *W3C Candidate Recommendation*, 2003.

[18] Chris Beckmann and Anind Dey. Siteview: Tangibly programming active environments with predictive visualization. In *Adjunct Proceedings of the Fifth Annual Conference on Ubiquitous Computing (UbiComp)*, pages 167–168, Seattle, Washington, USA, 2003.

[19] Marek Bell, Malcolm Hall, Matthew Chalmers, Phil Gray, and Barry Brown. Domino: Exploring Mobile Collaborative Software Adaptation. *Lecture Notes in Computer Science*, 2006.

[20] Victoria Bellotti and Keith Edwards. Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human Computer Interaction*, 16:193–212, 2001.

[21] David Benyon. Adaptive systems: A solution to usability problems. *User Modeling and User-Adapted Interaction*, 3(1):65–87, 1993.

[22] Jeremy Bernstein. A discussion of NATO+3d modular. *http://www.bootsquad.com/old_site/nato/nato00.html*, 2000.

[23] Meera M. Blattner, Denise A. Sumikawa, and Robert M. Greenberg. Earcons and icons: Their structure and common design principles. *Human-Computer Interaction*, 4(1):11–44, 1989.

[24] Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Uncertainty in Artificial Intelligence*, page 89âĂŞ97, 2003.

[25] John Boyle. A visual environment for the manipulation and integration of JAVA beans. *Bioinformatics*, 14(8):739–748, 1998.

[26] Stephen A. Brewster. Non-speech auditory output. In *The Human Computer Interaction Handbook*, pages 220–239. Lawrence Erlaum, Mahwah, NJ, 2002.

[27] British Broadcasting Corporation. Personal touch to Google homepage. *http://news.bbc.co.uk/1/hi/technology/6611571.stm*, 2007.

[28] Matt Bronstad, Kyle Lewis, and John Slatin. Conveying contextual information using non-speech audio cues reduces workload. In *Technology and Persons with Disabilities Conference*, 2003.

[29] Nat Brown and Charlie Kindel. Distributed Component Object Model Protocol–DCOM/1.0. *Internet Engineering Task Force (IETF), draft-brown-dcom-v1-spec-00*, 1998.

[30] Declan Butler. Virtual globes: The web-wide world. *Nature*, 439(7078):776–778, 2006.

[31] Muffy Calder, Phil Gray, and Chris Unsworth. Tightly coupled verification of pervasive systems. In *Proceedings of the Third International Workshop on Formal Methods in Interactive Systems (FMIS 2009). Electronic Communications of the EASST.*, volume 22, pages 32–48, Eindhoven, NL, 2010.

[32] Gaëlle Calvary, Joëlle Coutaz, Olfa Dâassi, Lionel Balme, and Alexandre Demeure. Towards a new generation of widgets for supporting software plasticity: the "Comet". In *Preproceedings of Engineering for Human-Computer Interaction / Design, Specification and Verification of Interactive Systems (EHCI/DSV-IS)*, volume 4, pages 41–60, Hamburg, Germany, 2004.

[33] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[34] Matthew Chalmers. A Population Approach to Ubicomp System Design. In *Proceedings of ACM-BCS Visions of Computer Science*, 2010.

[35] Matthew Chalmers and Ian MacColl. Seamful and Seamless Design in Ubiquitous Computing. *Workshop At the Crossroads: The Interaction of HCI and Systems Issues in the Fifth Annual Conference on Ubiquitous Computing (UbiComp)*, 2003.

[36] Julia S. Clark and Marilyn McGee-Lennon. A Stakeholder Centered Exploration of the Current Barriers to the Uptake of Home Care Technology in the UK. Technical Report TR-2009-314, Department of Computing Science, University of Glasgow, 2009.

[37] Cloanto Corporation. Specification of INI Files. *http://www.cloanto.com/specs/ini.html*, 2007.

[38] CNN International. Fortune 500. *http://money.cnn.com/magazines/fortune/fortune500/2007/full_list/index.html*, 2007.

[39] Herbert A. Colle and Alan Welsh. Acoustic masking in primary memory. *Journal of Verbal Learning & Verbal Behavior. Vol*, 15(1):17–31, 1976.

[40] Kay Connelly and Ashraf Khalil. Towards Automatic Device Configuration in Smart Environments. In *Proceedings of System Support for Ubiquitous Computing Workshop (UbiSys)*, Seattle, Washington, 2003.

[41] Andrew R. A. Conway, Michael J. Kane, Michael F. Bunting, D. Zach Hambrick, Oliver Wilhelm, and Randall W. Engle. Working memory span tasks: A methodological review and user's guide. . *Psychonomic Bulletin & Review*, 12:769–786, 2005.

[42] Tom Copeland. *Generating parsers with JavaCC*. Centennial Books, Alexandria, VA, ISBN 0-9762214-3-8, 2007.

[43] Stephen Crane, Naranker Dulay, H. Fosså, Jeff Kramer, Jeff Magee, Morris Sloman, and Kevin Twidle. Configuration management for distributed software services. In *Integrated Network Management IV: Proceedings of the Fourth International*

Symposium on Integrated Network Management, pages 29–42, Santa Barbara, CA, USA, 1995.

[44] Bridgette Craney. Handcrafted furniture: a matter of art and economics. *http://download.scientificcommons.org/48353*, 2002.

[45] John Crawford, Geoff Smith, Elizabeth Maylor, Sergio Della Sala, and Robert Logie. The Prospective and Retrospective Memory Questionnaire (PRMQ): Normative data and latent structure in a large non-clinical sample. *Memory*, 11(3):261–275, 2003.

[46] Murray Crease, Stephen A. Brewster, and Philip Gray. Caring, Sharing Widgets: A Toolkit of Sensitive Widgets. In *Proceedings of BCS Human-Computer Interaction (HCI'2000)*, pages 257–270, Sunderland, UK, 2000.

[47] Gianpaolo Cugola and Gian Pietro Picco. REDS: A Reconfigurable Dispatching System. In *Proceedings of the 6th international workshop on Software engineering and middleware*, pages 9–16, Portland, Oregon, USA, 2006.

[48] Yi Cui and Klara Nahrstedt. QoS-aware dependency management for component-based systems. In *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, pages 127–138, San Francisco, CA, 2001.

[49] Charles R. Darwin. *The Origin of Species by Means of Natural Selection. Or the Preservation of Favoured Races in the Struggle for Life*. Adamant Media Corporation, 1859.

[50] Stan Davis. From future perfect: Mass customizing. *Planning Review*, 17(2):16–21, 1989.

[51] Fausto J Sainz de Salces, David England, and David Llewellyn-Jones. Designing for all in the house. In *Proceedings of the 2005 Latin American conference on Human-computer interaction (CLIHC)*, pages 283–288, New York, NY, USA, 2005.

[52] Fausto Sainz de Salces, David England, and Paul Vickers. Household appliances control device for the elderly. In *Proceedings of the International Conference on Auditory Display (ICAD)*, Boston, MA, USA, 2003.

[53] Saeed Dehnadi and Richard Bornat. The camel has two humps. *Little PPIG (Psychology of Programming Interest Group) Workshop*, 2006.

[54] Micheal Dell. The power of virtual integration: an interview with Dell Computer's Michael Dell. Interview by Joan Magretta. *Harvard Business Review*, 76(2):73, 1998.

[55] Liam S. Docherty. *An Ontology Based Approach Towards A Universal Description Framework for Home Networks*. PhD Thesis, University of Stirling, Stirling, UK, 2009.

[56] Kevin Doughty, Keith Cameron, and Paul Garner. Three generations of telecare of the elderly. *Journal of Telemedicine and Telecare*, 2(2):71–80, 1996.

[57] Paul Dourish. Developing a Reflective Model of Collaborative Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(1):40–63, 1995.

[58] Paul Dourish. The appropriation of interactive technologies: Some lessons from placeless documents. *Computer Supported Cooperative Work (CSCW)*, 12(4):465–490, 2003.

[59] Troy Bryan Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc. Foster City, CA, USA, 1998.

[60] Emmanuel Dubois, Guillaume Gauffre, Cédric Bach, and Pascal Salembier. Participatory Design Meets Mixed Reality Design Models. *Computer-Aided Design Of User Interfaces (CADUI)*, pages 71–84, 2006.

[61] Emmanuel Dubois, Laurence Nigay, Jocelyne Troccaz, Olivier Chavanon, and Lionel Carrat. Classification space for augmented surgery, an augmented reality case study. In *Proceedings of Interact*, volume 99, pages 353–359, Edinburgh, UK, 1999.

[62] Marlon Dumas and Arthur H. M. ter Hofstede. UML activity diagrams as a workflow specification language. In *Proceedings of the 4th Int. Conference on the Unified Modeling Language (UML)*, volume 2185, pages 76–90, Toronto, Ontario, Canada, 2001.

[63] Robert Eckstein, David Collier-Brown, and Peter Kelly. *Using Samba*. O'Reilly, 1999.

[64] Eclipse Foundation. Equinox OSGi Release 4 (Equinox), 2009.

[65] W. K. Edwards, M. W. Newman, and J. Z. Sedivy. The Case for Recombinant Computing. Technical Report CSL-01-1, Xerox Palo Alto Research Center, 2001.

[66] W. Keith Edwards, Victory Bellotti, Anind K. Dey, and Mark W. Newman. Stuck in the Middle: Bridging the Gap Between Design, Evaluation, and Middleware. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, Fort Lauderdale, FL, USA, 2003.

[67] W. Keith Edwards, Mark W. Newman, Jana Sedivy, Trevor Smith, and Shahram Izadi. Challenge: Recombinant Computing and the Speakeasy Approach. In *The 8th Annual*

International Conference on Mobile Computing (MOBICOM), pages 279–286, Atlanta, Georgia, USA, 2002.

[68] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, volume 2919/2004, pages 333–336. LNCS, 2004.

[69] Eidgenossische Technische Hochschule Zurich. Concierge OSGi Release 3 (Concierge), 2009.

[70] E. Allen Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, 8:995–1072, 1990.

[71] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

[72] Patrick T. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):131, 2003.

[73] Kenneth Feldt. Programming Firefox: Building rich internet applications with XUL. *O'Reilly Media, Inc.*, 2007.

[74] Stephen Fickas. Clinical Requirements Engineering. In *Proceedings of the 27th International Conference on Software engineering (ICSE)*, pages 140–147, St. Louis, Missouri, USA, 2005.

[75] Klaus Finkenzeller. *RFID Handbook, Fundamentals and Applications in Contactless Smart Cards and Identification*. Wiley & Sons LTD April, 2003.

[76] William F. Finzer and Laura Gould. Rehearsal World: Programming by Rehearsal. *Byte*, 9(6), 1984.

[77] Gerhard Fischer, Raymond McCall, and Anders Morch. Design environments for constructive and argumentative design. *Special Interest Group on Computer Human Interaction (SIGCHI) Bulletin*, 20(SI):269–275, 1989.

[78] Marshall L. Fisher. What is the right supply chain for your product. *Harvard Business Review*, 75(2):105–116, 1997.

[79] Murielle Florins. *Graceful degradation: a method for designing multiplatform graphical user interfaces*. PhD Thesis, Université catholique de Louvain, Faculté des sciences economiques, sociales et politiques, 2006.

[80] Murielle Florins and Jean Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proceedings of the 9th international*

conference on Intelligent user interfaces, pages 140–147, Funchal, Madeira, Portugal, 2004.

[81] N. Freed. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. *RFC 2046 (http://www.ietf.org/rfc/rfc2046.txt)*, 1996.

[82] Stuart Friedberg. Transparent Reconfiguration requires a Third-Party Connect. *TR220, Computer Science Department, University of Rochester, New York, Nov*, 1987.

[83] Peter Frölich. Dealing with system response times in interactive speech applications. In *Computer Human Interaction (CHI) extended abstracts on Human factors in computing systems, ACM SIGCHI*, pages 1379–1382, Portland, OR, USA, 2005.

[84] Archana Ganapathi, Yi-Min Wang, Ni Lao, and Ji-Rong Wen. Why PCs are fragile and what we can do about it: a study of Windows registry problems. In *International Conference on Dependable Systems and Networks*, pages 561–566, Florence, Italy, 2004.

[85] Sebastian Garde and Petra Knaup. Requirements engineering in health care: the example of chemotherapy planning in paediatric oncology. In *Requirements Engineering*, volume 11, pages 265–278. Springer London, 2006.

[86] Rebecca Gardyn. Swap meet. *American Demographics*, 23(7):50–55, 2001.

[87] Guillaume Gauffre, Emmanuel Dubois, and Remi Bastide. Domain-Specific Methods and Tools for the Design of Advanced Interactive Techniques. *Models in Software Engineering*, pages 65–76, 2008.

[88] William W. Gaver. The SonicFinder: An interface that uses auditory icons. *Human-Computer Interaction*, 4(1):67–94, 1989.

[89] Jörg Geiβler. Shuffle, throw or take it! Working efficiently with an interactive wall. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, volume 98, pages 265–266, Los Angeles, CA, USA, 1998.

[90] Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41(4):587–601, 1973.

[91] Barney G. Glaser and Anselm L. Strauss. *The discovery of grounded theory: strategies for qualitative research*. Aldine, Chicago, 1967.

[92] GNOME Foundation. GConf configuration system. *http://www.gnome.org/projects/gconf/index.html*, 2007.

[93] Yaron Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. Simple service discovery protocol. *Internet Engineering Task Force (IETF), Draft draft-cai-ssdp-v1-03*, 1999.

[94] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[95] Google Inc. Google Gadgets. *http://www.google.com/apis/gadgets/index.html*, 2007.

[96] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java (TM) Language Specification* . Addison-Wesley Professional, 2005.

[97] T. C. Nicholas Graham, Catherine A. Morton, and Tore Urnes. ClockWorks: Visual Programming of Component-Based Software Architectures. *Journal of Visual Languages and Computing*, 7(2):175–196, 1996.

[98] Nicola M. Gray, Linda Sharp, Seonaidh C. Cotton, Mark Avis, Zoe Philips, Ian Russell, Leslie G. Walker, David Whynes, and Julian Little. Developing a questionnaire to measure the psychosocial impact of an abnormal cervical smear result and its subsequent management: the TOMBOLA (Trial Of Management of Borderline and Other Low-grade Abnormal smears) trial. *Quality of Life Research*, 14(4):1553–1562, 2005.

[99] Philip Gray, Tony McBryan, Chris Martin, Nubia Gil, Maria Wolters, Neil Mayo, Ken Turner, Liam Docherty, Feng Wang, and Mario Kolberg. A Scalable Home Care System Infrastructure Supporting Domiciliary Care. Technical Report CSM-173, Department of Computing Science and Mathematics, University of Stirling, UK, 2007.

[100] Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 209–218, Orlando, Florida, 2001.

[101] Donatien Grolaux, Peter Van Roy, and Jean Vanderdonckt. FlexClock, a Plastic Clock Written in Oz with the QTk toolkit. In *Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design table of contents*, pages 135–142, Vienna, Austria, 2002.

[102] Duelev P. Guelev, Mark Ryan, and Pierre Yves Schobbens. Model-checking access control policies. *Information Security*, pages 219–230, 2004.

[103] Christopher W. L. Hart. Mass customization: conceptual underpinnings, opportunities and limits. *International Journal of Service Industry Management*, 6(2):36–45, 1995.

[104] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[105] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, 37:29, 1972.

[106] H. Rex Hartson, Antonio C. Siochi, and Deborah Hix. The UAN: a user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems (TOIS)*, 8(3):181–203, 1990.

[107] Christine Hofmeister, Elizabeth White, and James Purtilo. Surgeon: a packager for dynamically reconfigurable distributed applications. *Software Engineering Journal*, 8(2):95–101, 1993.

[108] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley Publishing Company, 2004.

[109] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Åkesson, Boriana Koleva, Tom Rodden, and Pär Hansson. Playing with the Bits-User-configuration of Ubiquitous Domestic Environments. In *Proceedings of the Fifth Annual Conference on Ubiquitous Computing (UbiComp)*, pages 12–15, Seattle, Washington, USA, 2003.

[110] Hilary Hutchinson, Wendy Mackay, Bosse Westerlund, Benjamin B. Bederson, Allison Druin, Catherine Plaisant, Michel Beaudouin-Lafon, Stéphane Conversy, Helen Evans, Heiko Hansen, Nicolas Roussel, Björn Eiderbäck, Sinna Lindquist, and Yngve Sundblad. Technology probes: inspiring design for and with families. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 17–24, Ft. Lauderdale, Florida, USA, 2003.

[111] Elizabeth A. Inglis, Andrea Szymkowiak, Peter Gregor, Alan F. Newell, Nick Hine, Barbara A. Wilson, Jonathan Evans, and Praveen Shah. Usable technology? Challenges in designing a memory aid with current electronic devices. *Technology in Cognitive Rehabilitation*, page 77, 2004.

[112] R. Nicholas Jackiw and William F. Finzer. The geometer's sketchpad: Programming by geometry. In Allen Cypher, editor, *Watch What I Do: Programming by*

Demonstration. The MIT Press, 1998.

[113] Victor Lopez Jaquero, J. Vanderdonckt, F. Montero, and P. Gonzalez. Towards an Extended Model of User Interface Adaptation: the ISATINE framework. In *Engineering Interactive Systems (EIS)*, Salamanca, Spain, 2007.

[114] Rob Jarrett and Philip Su. *Building Tablet PC Applications*. Microsoft Press, 2002.

[115] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65, San Jose, California, 2007.

[116] Bonnie E. John and David E. Kieras. The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):320–351, 1996.

[117] Dylan M. Jones, Clare Madden, and Chris Miles. Privileged access by irrelevant speech to short-term memory: The role of changing state. *Quarterly Journal of Experimental Psychology: Human Experimental Psychology*, 44(4):645–669, 1992.

[118] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer Magazine*, 36(1):41–50, 2003.

[119] Usman Khan. *End User Programming of Home Care Applications*. Masters Thesis, University of Glasgow, 2008.

[120] Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD Thesis, University of Illinois at Urbana-Champaign, 2000.

[121] Suresh Kotha. Mass Customization: Implementing the Emerging Paradigm for Competitive Advantage. *Strategic Management Journal*, 16:21–42, 1995.

[122] David Kurlander. Chimera: Example-based graphical editing. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1998.

[123] Choonhwa Lee and Sumi Helal. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research*, 11(1):1–12, 2002.

[124] Henry Lieberman. Mondrian: A teachable graphical editor. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1998.

[125] Henry Lieberman. Tinker: A programming by demonstration system for beginning programmers. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1998.

[126] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[127] Lorna Lines and Kate S. Hone. Older Adults' Comprehension and Evaluation of Speech as Alarm System Output Within the Domestic Environment. In *2nd International Conference on Universal Access in Humahn Computer Interaction*, Crete, Greece, 2003.

[128] Lorna Lines and Kate S. Hone. Eliciting user requirements with older adults: lessons from the design of an interactive domestic alarm system. *Universal Access in the Information Society*, 3(2):141–148, 2004.

[129] Xiaoqing Liu, Chandra Sekhar Veera, Yan Sun, Kunio Noquchi, and Yuji Kyoya. Priority assessment of software requirements from multiple perspectives. In *28th Annual International Computer Software and Applications Conference*, Hong Kong, 2004.

[130] Bonnie MacKay, Carolyn Watters, and Jack Duffy. Web Page Transformation When Switching Devices. In *Proceedings of Sixth International Conference on Human Computer Interaction with Mobile Devices and Services (Mobile HCI'04), LNCS*, volume 3160, Glasgow, UK, 2004.

[131] Wendy E. Mackay. Patterns of sharing customizable software. In *Proceedings of the ACM conference on Computer Supported Cooperative Work (CSCW)*, pages 209–221, Los Angeles, California, United States, 1990.

[132] Wendy E. Mackay. *Triggers and barriers to customizing software*. ACM Press New York, NY, USA, 1991.

[133] Allan MacLean, Kathleen Carter, Lennart Lovstrand, and Thomas Moran. User-tailorable systems: pressing the issues with buttons. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 175–182, Seattle, Washington,USA, 1990.

[134] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying Distributed Software Architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, Barcelona, Spain, 1995.

[135] Makewave AB. Knopflerfish OSGi Release 4 (Knopflerfish 2), 2007.

[136] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph

processing. In *Proceedings of the 2010 international conference on Management of data*, Indianapolis, Indiana, USA, 2010.

[137] Lynn Margulis. *Symbiotic Planet: A New Look at Evolution*. Basic Books, 1998.

[138] Dave Marples and Peter Kriens. The Open Services Gateway Initiative: An Introductory Overview. *Communications Magazine, IEEE*, 39(12):110–114, 2001.

[139] D Masson, A Demeure, and G Calvary. Magellan, an Evolutionary System to Foster User Interface Design Creativity. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, 2010.

[140] Dimitri Masson. *Genetic Algorithm for Creativity Enhancement in UI design*. Masters Thesis, Laboratoire Infomatique de Grenoble, Grenoble, France, 2010.

[141] Claire Maternaghan and Ken Turner. A Component Framework for Telecare and Home Automation. In *7th Annual IEEE Consumer Communications & Networking Conference*, Las Vegas, Nevada, USA, 2010.

[142] David Maulsby and Ian H. Witten. Metamouse: An instructible agent for programming by demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1998.

[143] Tony McBryan and Phil Gray. A Generic Approach to the Evolution of Interaction in Ubiquitous and Context-Aware Systems. Technical Report TR-2007-260, Department of Computing Science, University of Glasgow, 2007.

[144] Tony McBryan and Phil Gray. A Model-Based Approach to Supporting Configuration in Ubiquitous Systems. In *Design, Specification and Verification of Interactive Systems 2008*, Kingston, Ontario, Canada, 2008.

[145] Tony McBryan and Phil Gray. A Framework for Runtime Evaluation, Selection and Creation of Interaction Objects (Poster) . In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, CMU, Pittsburgh, PA, USA, 2009.

[146] Tony McBryan and Phil Gray. User Configuration of Activity Awareness. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Salamanca, Spain, 2009.

[147] Tony McBryan and Phil Gray. Using Activity Awareness as a Run-time Interaction Configuration Testbed (Poster). In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*, CMU, Pittsburgh, PA, USA, 2009.

[148] Tony McBryan, Marilyn R McGee-Lennon, and Phil Gray. An Integrated Approach to Supporting Interaction Evolution in Home Care Systems. In *1st International Conference on Pervasive Technologies Related to Assistive Environments (PETRA)*, Athens, Greece, 2008.

[149] Marilyn McGee-Lennon and Julia S. Clark. Multi-Stakeholder Requirements in Home Care Technology Design. In *Workshop on Distributed Participatory Design. Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, Florence, Italy, 2008.

[150] Marilyn McGee-Lennon and Phil Gray. Addressing Stakeholder Conflict in Home Care Systems. In *British HCI Workshop on HCI, The Web and The Elderly*, Queen Mary, University of London, 2006.

[151] Marilyn McGee-Lennon and Phil Gray. Addressing the Challenges of Stakeholder Conflict in Home Care Systems. In *Workshop on Software Engineering Challenges for Ubiquitous Computing*, Lancaster, UK, 2006.

[152] Marilyn McGee-Lennon, Maria Wolters, and Tony McBryan. Audio Reminders in the Home Environment. In *Proceedings of the International Conference on Auditory Display (ICAD)*, Montreal, Canada, 2007.

[153] Marilyn R McGee-Lennon and Phil Gray. Including Stakeholders in the Design of Homecare Systems: Identification and Categorization of Complex User Requirements. In *Include Conference*, Royal College of Art, London, 2007.

[154] M. Douglas McIlroy. Mass produced software components. *Software Engineering Concepts and Techniques*, pages 88–98, 1969.

[155] Microsoft Corporation. The Windows 98 Config.txt File. *http://support.microsoft.com/?kbid=232557*, 1998.

[156] Microsoft Corporation. .NET Configuration Namespace. *http://msdn2.microsoft.com/en-us/library/system.configuration.aspx*, 2007.

[157] Microsoft Corporation. Windows Vista: Compare Editions. *http://www.microsoft.com/windows/products/windowsvista/editions/choose.mspx*, 2007.

[158] Alister Morrison, Paul Tennent, and Matthew Chalmers. Coordinated visualisation of video and system log data. In *Proceedings of the Fourth International Conference on Coordinated & Multiple Views in Exploratory Visualization*, volume 6, pages 91–102, London, United Kingdom, 2006.

[159] Brad A. Myers. Text Formatting by Demonstration. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 251–256, New Orleans, Louisiana, USA, 1991.

[160] Eric Newcomer. *Understanding Web Services: XML, Wsdl, Soap, and UDDI*. Addison-Wesley Professional, 2002.

[161] Mark W. Newman, Ame Elliott, and Trevor F. Smith. Providing an integrated user experience of networked media, devices, and services through end-user composition. *Pervasive Computing*, pages 213–227, 2008.

[162] Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, and Trevor F. Smith. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 147–156, London, England, 2002.

[163] Blair Nonnecke and Jenny Preece. Lurker demographics: Counting the silent. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 73–80, The Hague, NL, 2000.

[164] Ulrich Norbisrath and Christof Mosler. Functionality configuration for eHome systems. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, 2006.

[165] Jon O'Brien and Tom Rodden. Interactive systems in domestic environments. In *Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 247–259, 1997.

[166] Department of Health. Our health, our care, our say: a new direction for community services, 2006.

[167] OSGI Alliance. *OSGI Service Platform, Release 3*. IOS Press, Inc., 2003.

[168] Eun Kyoung Paik, Minho K. Shin, Jaeryung Hwang, and Jaeyoung Choi. Design Goals and General Requirements for Future Network. *N13490, Korea Technology Center*, 2008.

[169] Fabio Paterno, Cristian Mancini, and Silvia Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (Interact)*, pages 362–369, Rio de Janeiro, Brazil, 1997.

[170] Perceptive Automation LLC. Indigo. *http://www.perceptiveautomation.com/*, 2010.

[171] Mark Perry, Alan Dowdall, Lorna Lines, and Kate Hone. Multimodal and ubiquitous computing systems: Supporting independent-living older users. *IEEE Transactions on Information Technology in Biomedicine*, 8(3):258–70, 2004.

[172] Carl Adam Petri. *Kommunikation mit Automaten*. PhD Thesis, Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ, 1962.

[173] Joelle Pineau, Michael Montemerlo, Martha Pollack, Nicholas Roy, and Sebastian Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, 2003.

[174] Satyan G. Pitroda. Electronic diary - Patent, United States 3999050, 1976.

[175] Tim Place and Trond Lossius. Jamoma: A Modular Standard for Structuring Patches in Max. In *Proceedings of International Computer Music Conference*, New Orleans, Louisiana, USA, 2006.

[176] Martha E. Pollack. Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI magazine*, 26(2):9, 2005.

[177] Martha E. Pollack, Laura Brown, Dirk Colbry, Colleen E. McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan, and Ioannis Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3):273–282, 2003.

[178] Catherine Pope, Sue Zieland, and Nicholas Mays. Qualitative research in health care. *BMJ*, 320:114–116, 2000.

[179] Nathaniel G. Pryce. *Component Interaction in Distributed Systems*. PhD Thesis, University of London, London, UK, 2000.

[180] Miller S. Puckette. Max/MSP. *http://www.cycling74.com/products/maxmsp*, 2006.

[181] Miller S. Puckette. Max/Jitter. *http://www.cycling74.com/products/jitter*, 2007.

[182] James M. Purtilo. The POLYLITH software bus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(1):151–174, 1994.

[183] Patrick Rabbitt, Mike Anderson, Ellen Bialystok, and Fergus I. Craik. The lacunae of loss? Aging and the differentiation of human abilities. *Lifespan Cognition: Mechanisms of Change*, 2005.

[184] Peter D. Rail. Configuration file management - Patent, United States 5740431, 1998.

[185] Paul Resnick and Hal R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[186] Yann Riche. *Designing Communication Appliances to Support Aging in Place*. PhD Thesis, Universite Paris-Sud, France, 2008.

[187] Yann Riche and Wendy Mackay. MarkerClock: A communicating augmented clock for elderly. In *Proceedings of Interact*, pages 408–411, Rio de Janeiro, Brasil, 2007.

[188] Jane Ritchie and Liz Spencer. Qualitative data analysis for applied policy research. In A. Bryman and R. Burgess, editors, *Analysing qualitative data*, volume 1993, pages 173–194. Routledge, London, 1993.

[189] Linda A. Roberts and Cynthia A. Sikora. Optimising feedback signals for multimedia devices: Earcons vs. Auditory icons vs. Speech. In *Proceedings of International Ergonomics Association (IEA)*, Tampere, Finland, 1997.

[190] Tom Rodden and Steve Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 9–16, Fort Lauderdale, Florida, USA, 2003.

[191] Dale Rogerson. *Inside COM: Microsoft's Component Object Model*. Microsoft Press Redmond, Washington, 1997.

[192] Pierre Salame and Alan D. Baddeley. Disruption of short-term memory by unattended speech: Implications for the structure of working memory. *Journal of Verbal Learning & Verbal Behavior. Vol*, 21(2):150–164, 1982.

[193] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-aware applications. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 431–441, Pittsburgh, Pennsylvania, USA, 1999.

[194] Timothy A. Salthouse, Renée L. Babcock, and Raymond J. Shaw. Effects of adult age on structural and operational capacities in working memory. *Psychology and Aging*, 6(1):118–127, 1991.

[195] Mark Allen Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[196] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[197] Nitin Sawhney and Chris Schmandt. Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(3):353–383, 2000.

[198] J. Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, 1999.

[199] Bill N. Schilit and Marvin M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.

[200] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[201] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content Based Routing with Elvin4. In *Enterprise Security, Enterprise Linux, Australian National University (AUUG2k)*, Canberra, Australia, 2000.

[202] Marcos Serrano, Laurence Nigay, Jean-Yves L. Lawson, Andrew Ramsay, Roderick Murray-Smith, and Sebastian Denef. The openinterface framework: a tool for multimodal interaction. In *Computer Human Interaction (CHI) extended abstracts on Human factors in computing systems, ACM SIGCHI*, pages 3501–3506, Florence, Italy, 2008.

[203] Helen Sharp, Anthony Finkelstein, and Galal Galal. Stakeholder identification in the requirements engineering process. In *Proceedings of 10th International Workshop on Database & Expert Systems Applications (DEXA)*, pages 387–391, Florence, Italy, 1999.

[204] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, 1995.

[205] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1996.

[206] Jon Siegel. *CORBA 3 Fundamentals and Programming with Cdrom*. John Wiley & Sons, Inc. New York, NY, USA, 1999.

[207] Cynthia A. Sikora, Linda Roberts, and La Tondra Murray. Musical vs. Real world feedback signals. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, pages 220–221, Denver, Colorado, 1995.

[208] Giovani Da Silveira, Denis Borenstein, and Flávio S. Fogliatto. Mass customization: Literature review and research directions. *International Journal of Production Economics*, 72(1):1–13, 2001.

[209] Jesper Simonsen and Finn Kensing. Using ethnography in contextural design. *Communications of the ACM*, 40(7):88, 1997.

[210] David Canfield Smith. Pygmalion: An executable electronic blackboard. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.

[211] Tony Smith. Dell overtakes Compaq (in US). *http://www.theregister.co.uk/2000/01/25/pc_sales_up_23_per/*, 2000.

[212] Michael Solomon, Gary Bamossy, Soren Askegaard, and Margaret K. Hogg. *Consumer behaviour: a European perspective*. Prentice Hall Europe, 1999.

[213] Joao Pedro Sousa and David Garlan. Improving User-Awareness by Factoring it Out of Applications. In *Proceedings of System Support for Ubiquitous Computing Workshop (UbiSys)*, Seattle, Washington, 2003.

[214] Anselm L. Strauss. *Qualitative analysis for social scientists*. Cambridge Univ Pr, 1987.

[215] Roy Suddaby. From the editors: What grounded theory is not. *Academy of Management Journal*, 49(4):633–642, 2006.

[216] Brad Templeton. Down with files that begin with a dot. *http://groups.google.co.uk/group/net.unix-wizards/browse_thread/thread/3110462c9dec6da1/eb0be2b2f44abe64?lnk=st&q=&rnum=4030&hl=en*, 1982.

[217] David Thevenin and Joëlle Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of Interact*, volume 99, pages 110–117, Edinburgh, UK, 1999.

[218] Stewart Thomson, John McCall, and David Crossen. Component Based Visual Software Engineering. In *Proceedings of the Second International Conference on Enterprise Information Systems*, page 363, Leeds, UK, 2000.

[219] Stephen Todd and Latham William. *Evolutionary art and computers*. Academic Pr, 1992.

[220] Tunstall Healthcare (UK) Ltd. Tunstall Telehealthcare Equipment. *http://www.tunstall.co.uk/*, 2010.

[221] David Ulph and Nir Vulkan. *E-commerce, Mass Customisation and Price Discrimination*. University of Bristol, Department of Economics, 2000.

[222] U.S. Census Bureau. Population Division, Interim State Population Projections, 2005.

[223] Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Bouillon, Daniela Trevisan, and Murielle Florins. USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In *W3C Workshop on Multimodal Interaction.* , pages 19–20, Sophia Antipolis, France, 2004.

[224] Wim Vanderperren and Bart Wydaeghe. Towards a new component composition process. In *Proceedings of the Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 322–329, Washington, DC, USA, 2001.

[225] John Veizades, Erik Guttman, Charles E. Perkins, and Scott Kaplan. Service Location Protocol. *RFC 2165 (http://www.ietf.org/rfc/rfc2165.txt*, 1997.

[226] Roman Vilimek and Thomas Hempel. Effects of speech and non-speech sounds on short-term memory and possible implications for in-vehicle use. In *Proceedings of the International Conference on Auditory Display (ICAD)*, Limerick, Ireland, 2005.

[227] Will Wade. Custom Stamps as Status Symbols. *http://www.nytimes.com/2007/05/25/business/media/25adco.html?ex=1183608000&en=55c8f57a283a856d&ei=5070*, 2007.

[228] ChuanJun Wang. *A Resident Activity Monitor for Homecare*. Masters thesis, University of Glasgow, 2007.

[229] Feng Wang, Liam S. Docherty, Kenneth J. Turner, Mario Kolberg, and Evan H. Magill. Service and Policies for Care at Home. In *International Conference on Pervasive Computing Technologies for Healthcare*, Innsbruck, Austria, 2006.

[230] Mark Weiser. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, 1999.

[231] Daniel S. Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically personalizing user interfaces. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[232] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, volume 18, pages 1173–1178, 2003.

[233] John Williamson, Rodderick Murray-Smith, and Stephen Hughes. Shoogle: excitatory multimodal interaction on mobile devices. In *Proceedings of Computer Human Interaction (CHI) conference on Human factors in computing systems, ACM SIGCHI*, San Jose, California, USA, 2007.

[234] Michael Wilson, Evan H. Magill, and Mario Kolberg. An Online Approach for the Service Interaction Problem in Home Automation. In *Proceedings of Consumer Communications and Networking Conference (CCNC)*, pages 251–256, Las Vegas, Nevada, 2005.

[235] Ian H. Witten and Dan Mo. TELS: Learning Text Editing Tasks from Examples. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1998.

[236] Patricia Wright, Nick Rogers, Christine Hall, Barbara Wilson, Jonathan Evans, Hazel Emslie, and Christine Bartram. Comparison of pocket-computer memory aids for people with brain injury. *Brain injury*, 15(9):787–800, 2001.

[237] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K.S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.

A

Glossary

Adaptive System: a system which is capable of changing its behaviour in response to an internal or external change. Introduced in Chapter 2.

Application Task: an application task is a task that implements some application logic that aims to achieve some users high level goal. i.e. Notify me when the temperature gets too low. Introduced in Section 7.2.

Channel: a named communication medium between services within the framework. An evaluation function can be assigned to a channel for a specific application task which will bind that channel to the results of an evaluation function execution (a set of possibilities). Introduced in Section 7.2.

Component: a functional unit within a system; a component is an endpoint in a possibility - usually representing a physical or software device which can communicate with the user. Introduced in Section 7.2.

Configuration: a collection of functional units, which may be connected, which is complete enough to fulfil some or all of the goals of the system. Introduced in Chapter 2.

Configuration Evaluation Function: The purpose of a configuration evaluation function is to rank, filter or otherwise analyse these possibilities to reduce them to a set of selected possibilities which represent a configuration decision that has been made. Referred to as

evaluation functions within the text. Introduced in Section 5.2.

Configuration Possibility: encapsulated solution (consisting of interaction components, techniques and devices) that can offer interaction between a system task and a user. Referred to as *possibilities* within the text. Introduced in Section 5.2.

Configuring/Reconfiguring: the selection (or reselection) of components, services or features to better suit the users needs or the requirements of the application. Introduced in Chapter 2.

Customisation: supplier driven configuration of a product within a fixed set of options. Introduced in Chapter 2.

Evolution: multiple related directed instances of reconfiguration. Introduced in Chapter 2.

Framework: the MATCH framework is a software middleware which implements and demonstrates the features discussed in this thesis. Introduced in Chapter 7.

Interaction Evolution: multiple related instances of interaction configuration (customisation or personalisation) over time that have a goal to change some aspect of the systems interaction behaviour. Introduced in Chapter 4.

Interaction Manager: a process which is responsible for coordinating evaluation function calls. Initially introduced in Section 6.6.1 and expanded upon in detail in Section 7.3.

Mass Customisation: customisation on a large scale. Introduced in Chapter 2.

Personalisation: user driven customisation where the user provides their own configuration options. Introduced in Chapter 2.

Possibility Graph: a directed graph of available components - from which the available possibilities can be derived. Introduced in Section 5.2.

Service: in the context of the implementation a Service is a task or a component within the framework which can be discovered and used within possibilities. Introduced in Section 7.2.

Subsystem: these refer to either mandatory or optional modular components that are deployed as OSGi bundles as part of the framework described in Chapter 7.

Task: a task is a software component that can be started or stopped in the framework to provide some functionality and may be involved in possibilities. An example is a speech synthesis task that converts text into audio. Introduced in Section 7.2.

B

Supplementary Materials

The following additional supplementary materials are available from the author on request.

- Source code for:

  - Audio Reminder Application as described in Chapter 3.

  - MATCH Framework as described in Chapter 7 and prototype applications. Note that this does not include components restricted by copyright license. Neither Cerevoice or the applications that were built by other developers within the MATCH framework can be provided by the author.

  - Activity Monitor applications as described in Chapter 8.

- Anonymised transcripts of interviews undertaken in Chapter 8.

C

User Manual - Evolutionary Configuration

MATCH Activity Monitor Help Guide

This is a short help guide that will help you to set up some common configurations in the Activity Monitor system.

The Activity Monitor system is used to direct a selection of different "input" devices to a selection of different "output" devices.

When you first start the Activity Monitor you will be greeted with a screen similar to the one below.
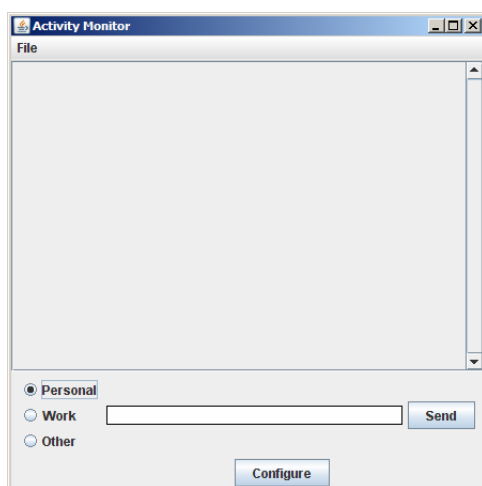
This is split into two sections. The local GUI is highlighted in RED and the activity entering section is highlighted in BLUE.
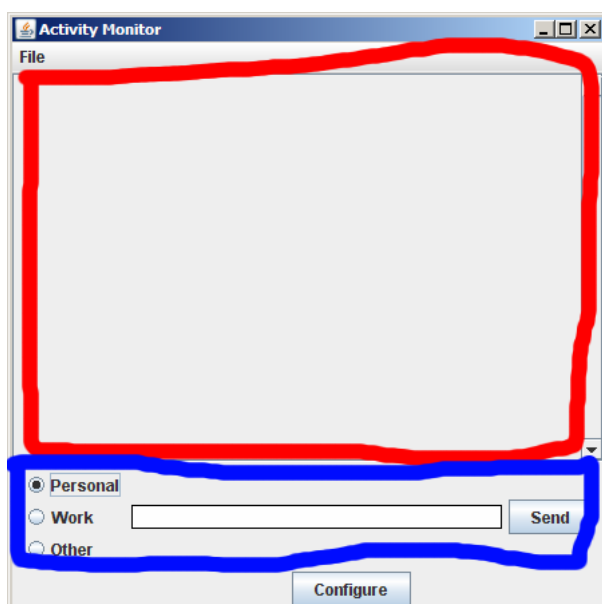
The GUI area will display any activities that have been directed to your local GUI in the Configure screen (we will come back to this shortly). The activity entering

section is used for you to manually type in messages to send to yourself or other people.

If you click the configure option (or select File, Configuration) then you will be presented with the following screen.
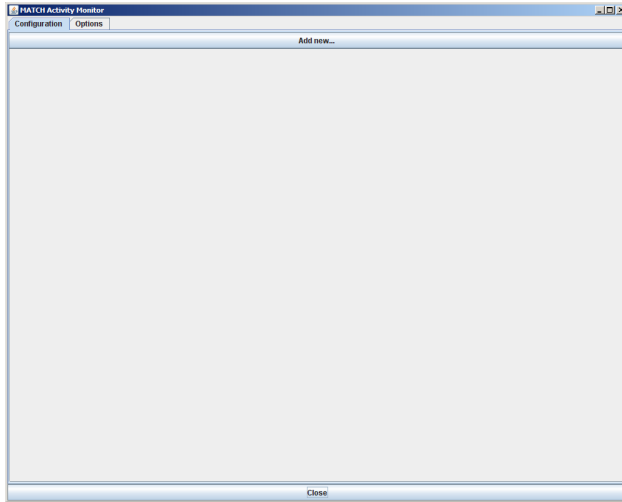
By default it will not have anything setup in it. But you can add a new "rule" by selecting the "Add new" button at the top of the screen.
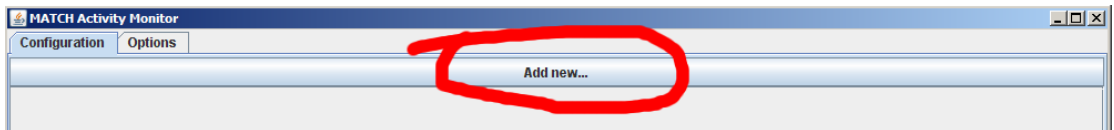
The interesting options this provides are highlighted below. These drop down boxes allow you to select what you want to go where. The dropdown box on the left selects INPUTS while the one on the right selects OUTPUTS.
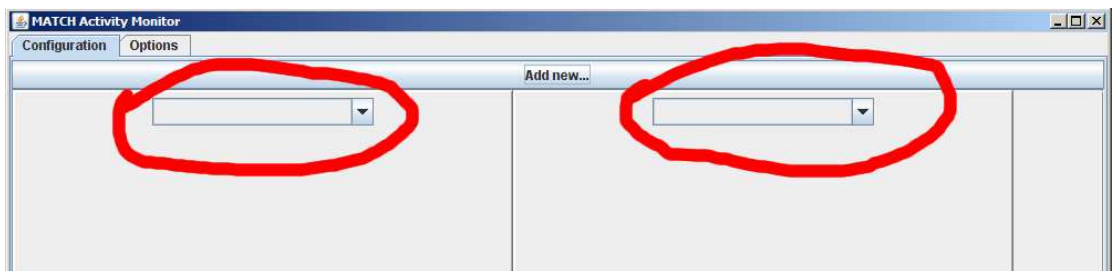
Example Activity Tasks

Task 1: Setting your own messages to be displayed on your GUI

Step one is to select the Devices option from the drop down box which allows you to select between all the local inputs available on your machine.
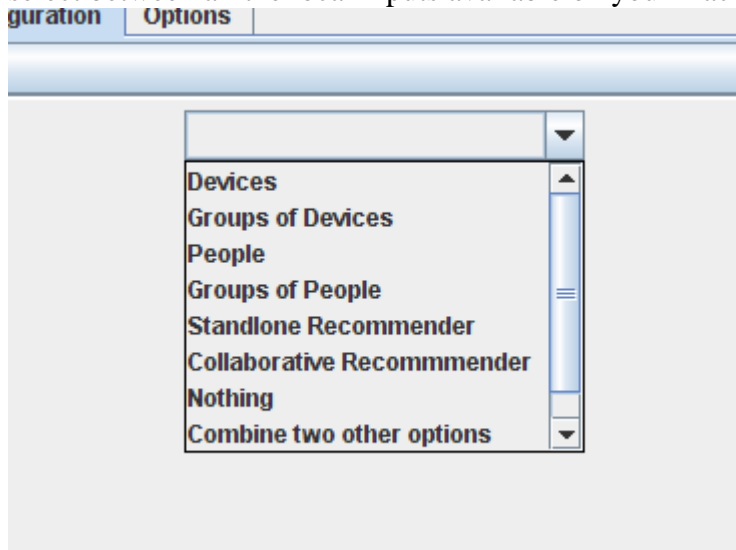

This will populate the list with a selection of devices. Check the checkboxes for Personal, Work and Other messages. These correlate to the 3 options you can select when typing in messages yourself.
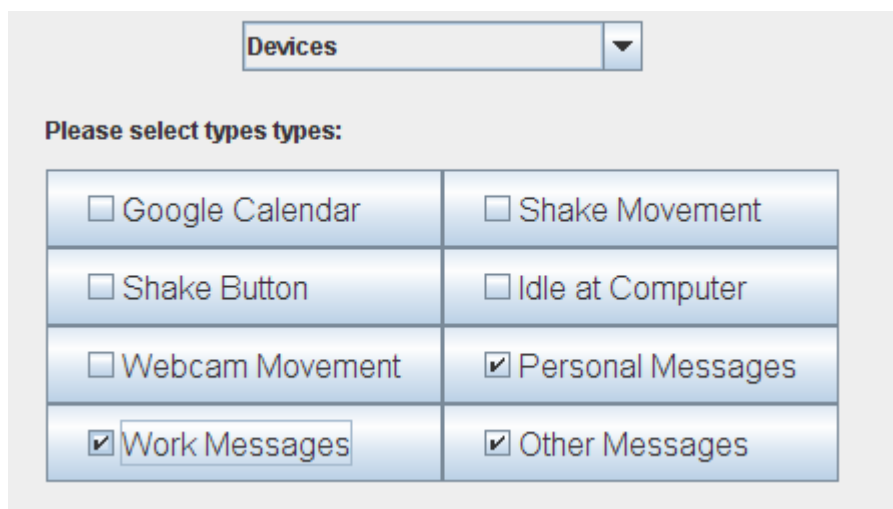
Repeat this on the right hand side and select the GUI. Your screen should now resemble the following.
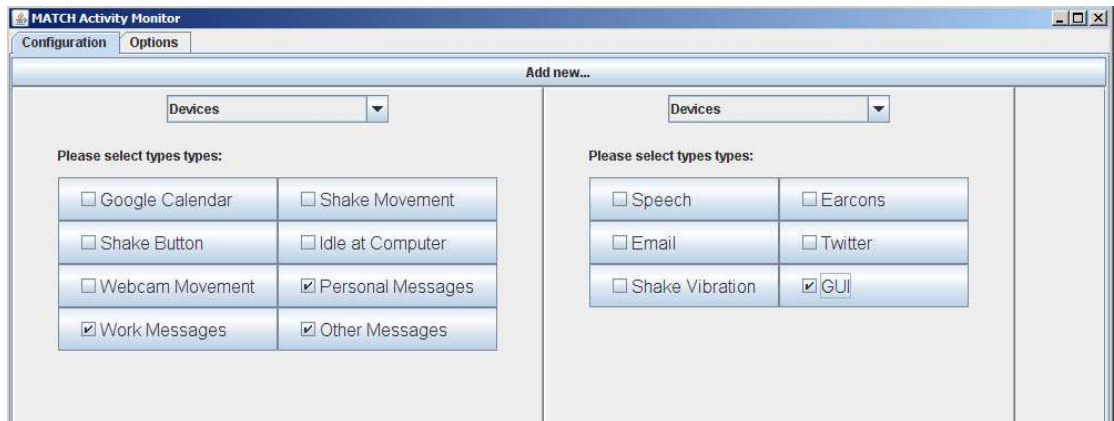
Close this window by selecting the Close option and then type a personal message into the text entry area and click send. This will then display the message on your GUI.
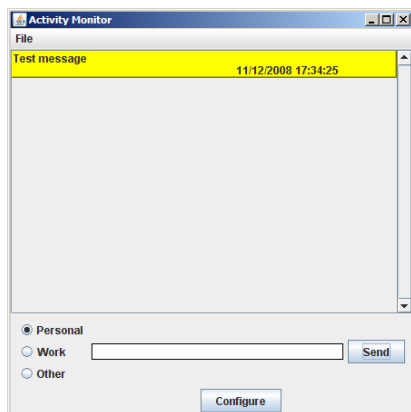
## Task 2: Sending your messages to other people

Click the Configure button again. And change the selection on the drop down to "Groups of People" for the output (right most) drop down box and select the "Everyone" option such that it looks like the following.

This will then send all your messages to everyone else. If anyone else is listening for your messages then they will receive them. Note that someone needs to have selected what to do with messages from you before they will receive them.
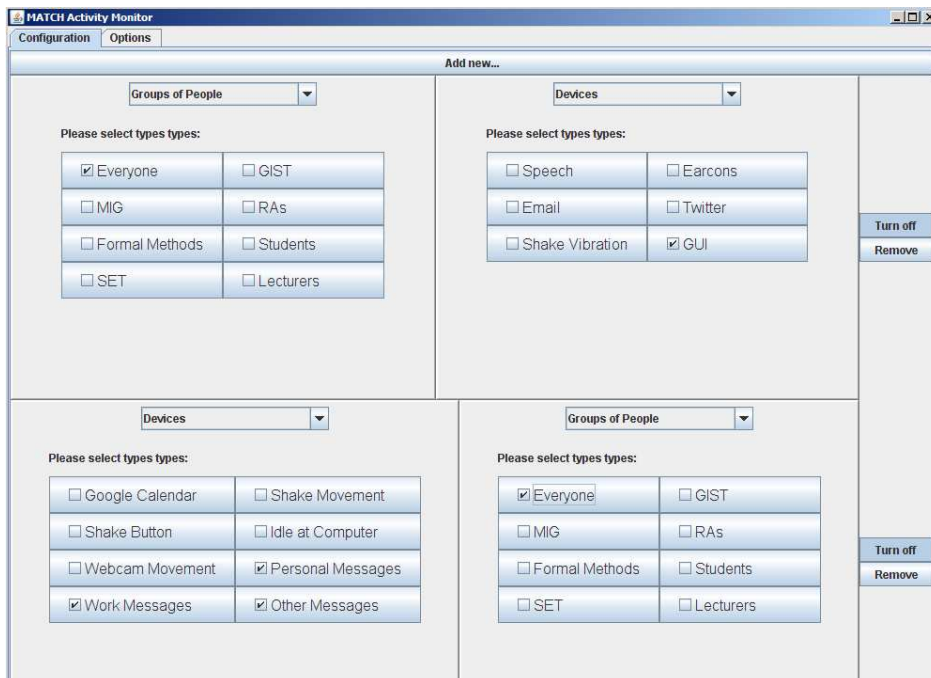
Task 3: Receiving messages from other people

Next you want to decide what you want to do with messages received from other people. Go back into the configuration screen and select the "Groups of People" option on the left hand side and select "Everyone". Select "Devices" from the drop down menu and select "GUI" on the right hand side. You will now receive any messages from everyone on your GUI.
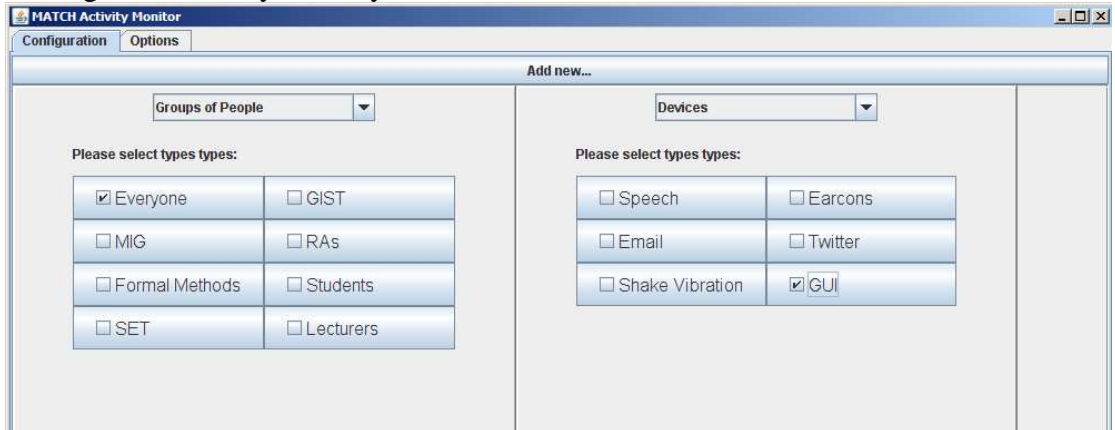
Task 4: Multiple Rules

In the previous examples we have only used a single rule at a time. But it is possible to setup multiple rules to obtain the desired behaviour. Simply click the "Add new" button to add as many rules as you require.


Shown above is a setup where all messages from other people are sent to your GUI while any messages you type in are sent to everyone.

Task 5: Combining two options

It is also possible to combine together two of the drop down options for the cases where you want to send an input to multiple places (or vice versa).

To do this select the "Combine two other options" entry in the dropdown box. When you first select the Combine two options menu then you will be presented with two further drop down boxes as shown below which you can then choose between as normal.
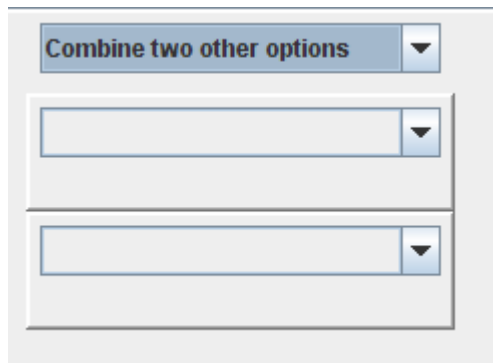
The rule presented below for example sends all personal, work and other messages to your local GUI so you can see them as well as sending them to everyone else.
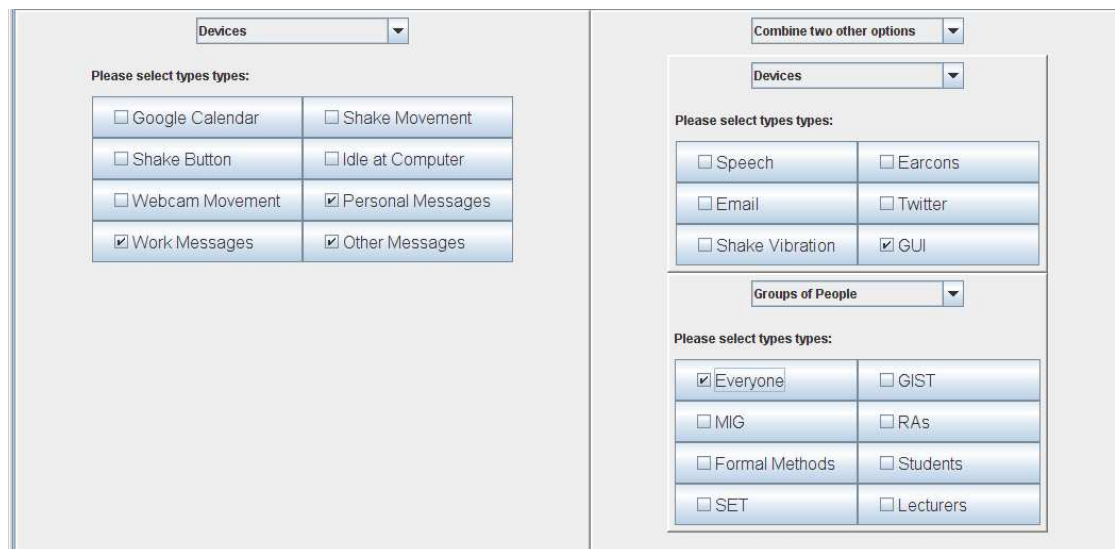
Task 6 : Context Sensitive

Another option similar to the combine option is to do different things in different circumstances. The system will monitor your idle status (i.e. how long you have been at your machine) and can send activity messages to different places depending on how long you have been away from your computer.
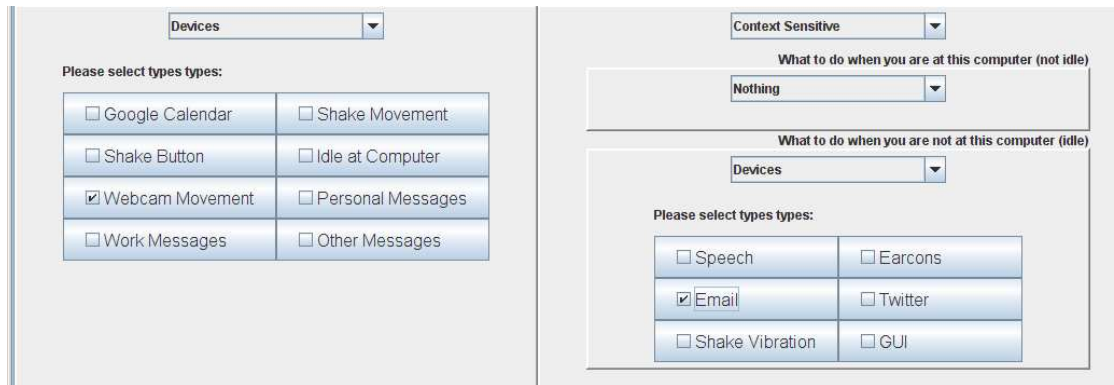
The above rule sends you an email when you have been away from your machine and webcam movement is detected. Acting as a simple security system.

Task 7: Using Recommenders

There are two recommender systems supplied which can automatically infer an appropriate choice of input or output for a rule based on the contents of the other side of the rule.
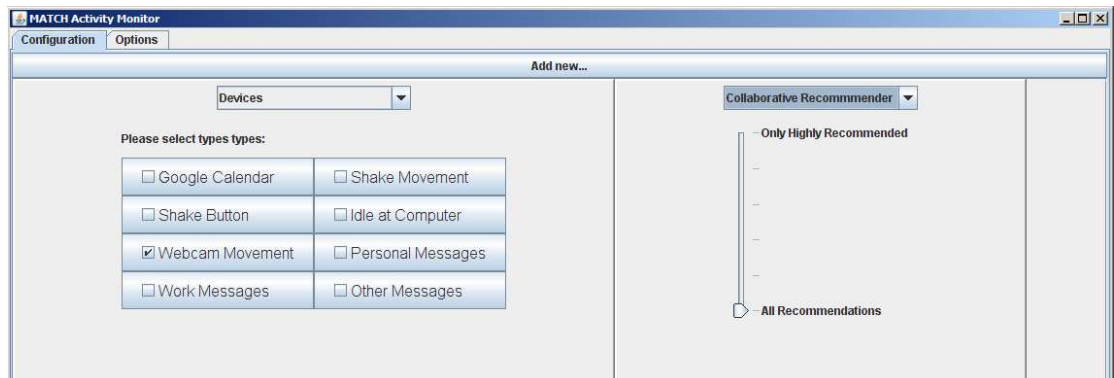
The above rule demonstrates the Collaborative Recommender which will automatically select an appropriate output device to send Webcam movement to based on what other people tend to send webcam movement messages to. There is also a standalone recommender that does the same job but based entirely on what you have done in the past. The Recommenders can be adjusted in sensitivity using the slider.

Note that setting both the input and output to a recommend will not work since it looks at the opposite option to determine what it should do.

Task 8: Adjusting Settings
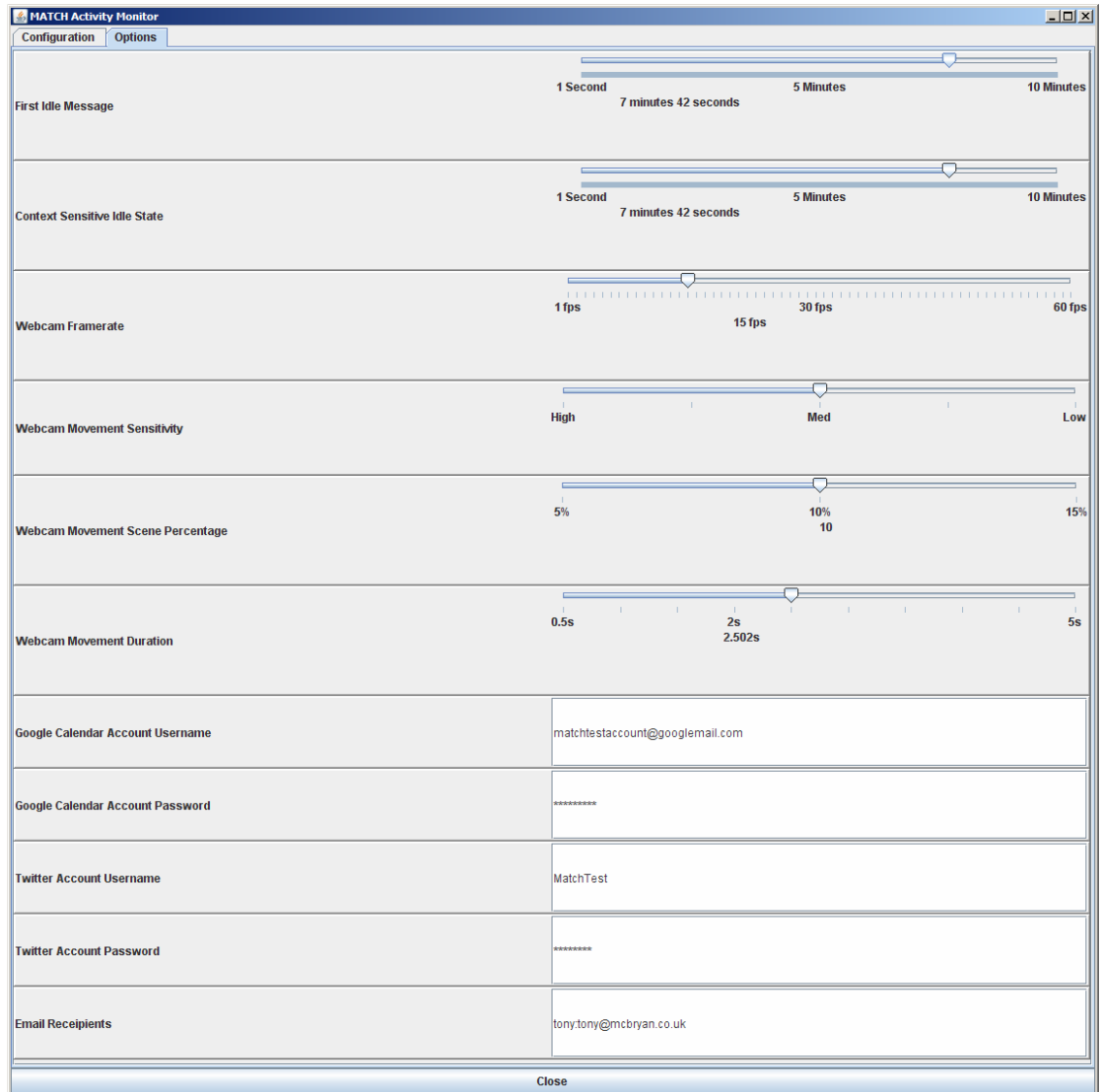
If you click the "Options" tab at the top of the configuration panel then you will be presented with global settings that alter the behaviour of the application.

The *first idle message* option dictates the period of time that must pass before messages about your Idle Status at the Computer will be sent.

The *context sensitive idle state* option dictates how long must pass before the context sensitive option regards you as being away from your machine and changes what is used.

The *webcam frame rate* is used to specify the frequency that the webcam will be queried to detect movement. Higher settings here consume more CPU power on the machine but increase the range of movements that can be detected. Setting a low value here will reduce CPU usage by the Activity Monitor but will result in some quick movements being missed.

The *webcam movement sensitivity* option selects how sensitive the webcam is to movement and the *webcam movement scene percentage* selects how much movement of the scene as seen by the camera needs to take place before it is counted as movement.

It may be necessary to adjust these values depending on what type of movement you intend on capturing.

Finally the options panel also includes the usernames and passwords for several services. If these are changed then a restart of the machine is required.

Task 9 : Turning off / removing rules

It is possible to temporarily turn off a rule or to remove a rule by selecting the "Turn Off" or "Remove" options next to a rule.
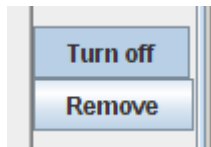
Components

List of components and what they do:

- Twitter (www.twitter.com) is a social networking application that allows you to post your activity messages to a webpage for others to view. Your Twitter page will be located at http://www.twitter.com/<twitter-username> i.e. If you username for Twitter is listed as "username" then your Twitter page will be http://www.twitter.com/username.

- Google Calendar allows you to have calendar appointments sent to yourself or other people. To access your calendar go to www.google.com and sign in using your Google Account username and password and select "More -> Calendar". If you are an Outlook user you can sync your Outlook appointments to Google Calendar (http://www.google.com/support/calendar/bin/answer.py?answer=89955)

- Some users will have been supplied with a SHAKE device which is capable of detecting shaking of the device as well as the button on the side. Additionally the SHAKE can be made to vibrate when an activity message arrives.

- The Idle at Computer option will send messages indicating you are no longer at your PC. These messages will continue while you are not at your PC but the duration between each message will double.
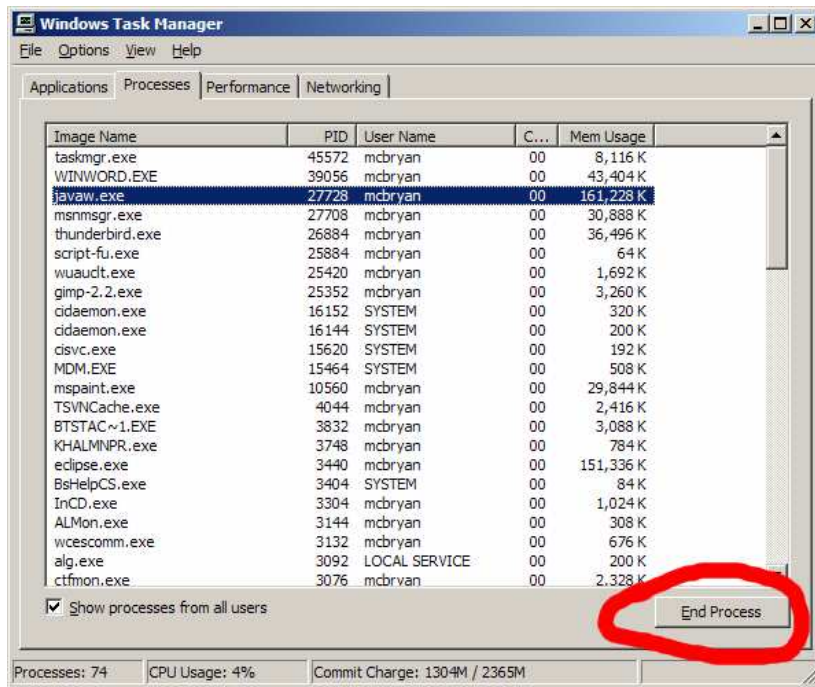
- The Webcam can detect Movement within the field of view of the camera frame and will send a message to indicate this whenever movement is detected.

- There is a speech synthesis application including which can verbalise the messages you receive. The exact text that will be spoken varies depending on the type of message.

- Earcons are auditory icons and are represented by different musical tones for different messages.

- Email will send a message to your email account as specified in the global settings panel. The format of email addresses is name:emailaddress;name:emailaddress. So if you wanted to send emails to yourself and another person it might look like this: me:myemail@domain.com;otherperson:theiremail@domain.com

- The final option available is to send messages to other people.

This is beta/research software so expect bugs.

As this is a prototype system there will likely be a few bugs existing. Generally resetting the system will cure these.

To reset the system simply select File, Exit, or click the red X in the top right corner of the application.

If this fails open the Task Manager application for your version of Windows, select the Processes tab and end the javaw.exe process.

Reset if:

* If it takes a while to startup.

* If it locks up.

* If something (speech etc stops working).

Incident report sheet

Please log impressions, problems and experiences with the software in this form.

When	Incident	Type of incident – Bug (please include severity), interesting observation,

D

User Manual - User Configuration Behaviour

Participant Information Sheet

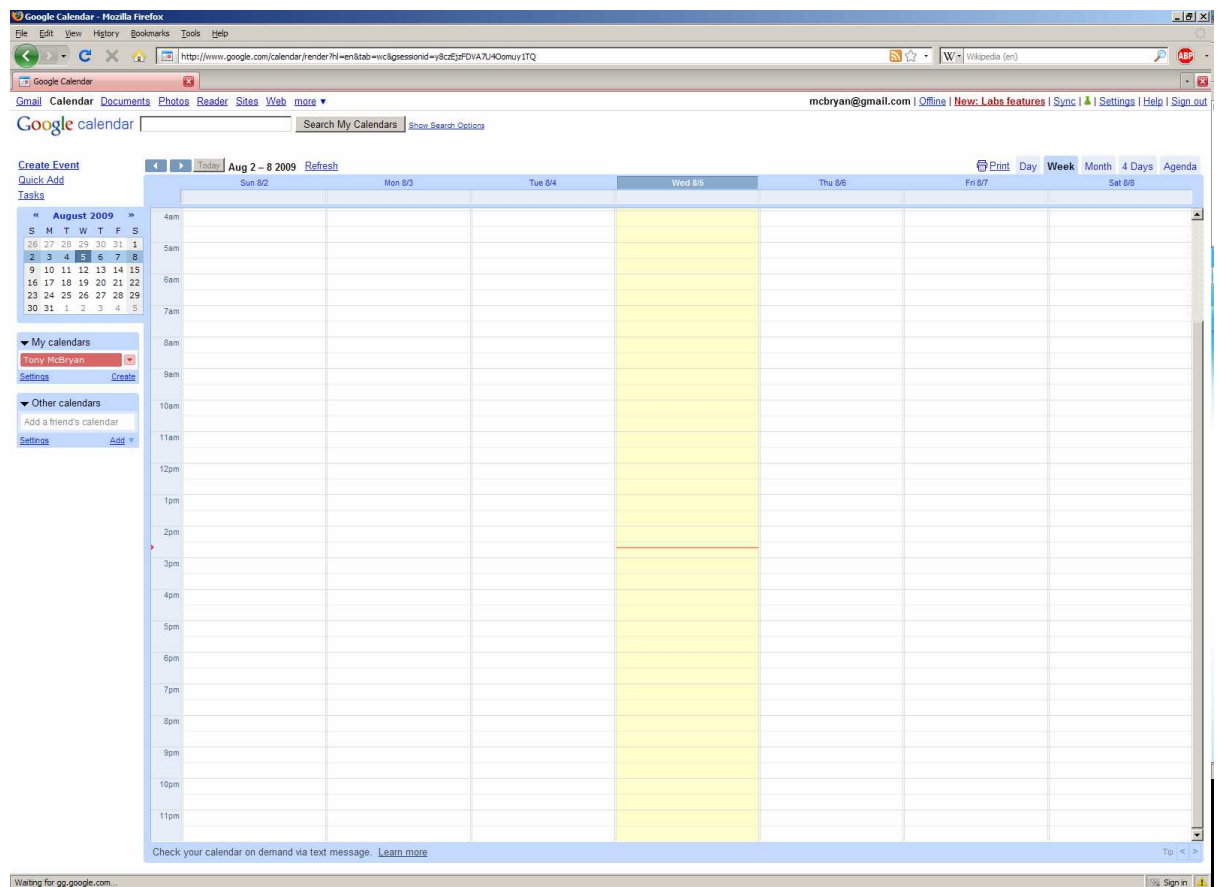## Calendar update

The calendar feature retrieves events to notify you about from Google Calendars.

In order to add events you must use the Google Calendar interface from a regular desktop PC at :

<center>

http://www.google.com/calendar

</center>

Username:

Password:

This will give you a display such as:

Just click and fill in the event details and click Create Event.
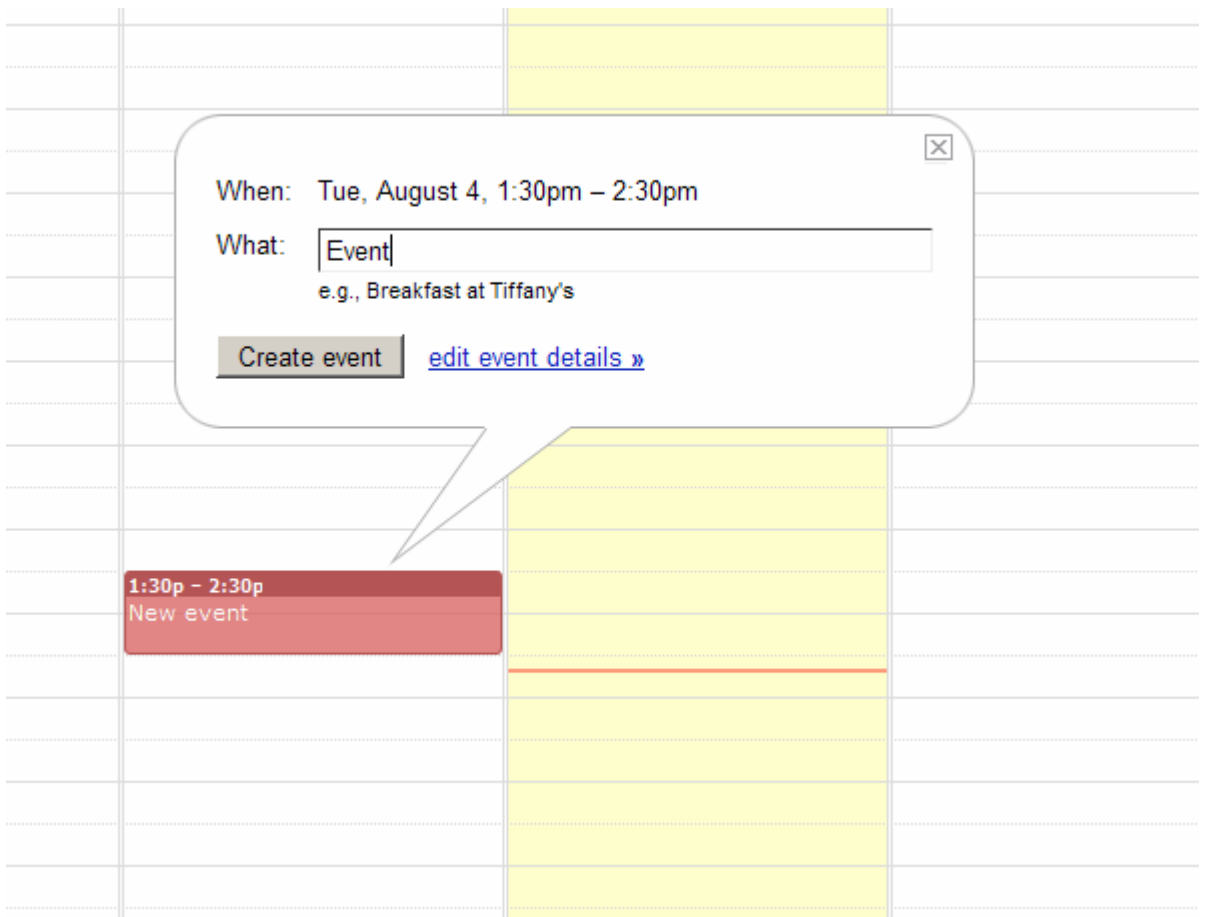
This will by default popup a notification on the UMPC ten minutes before the event is due to start (as long as the calendar as been connected to an output of course).

You can change the length of time before the notification by selecting the "edit event details" option and changing the "Reminder" option.
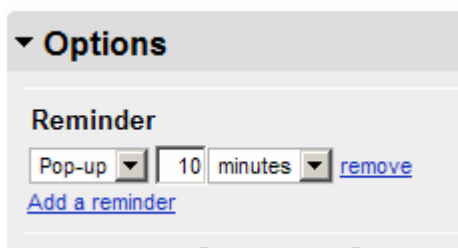
You can have multiple reminders for each event if you wish (select Add a reminder). Please leave reminders on the "Pop-up" setting.

Please set reminders on the Calendar at least 30 minutes before the event time to ensure that the UMPC can update in time.

Website Users

Along with yourself you can nominate a number of other users who you can communicate with via the internet. They just need to visit a special web page where they can receive any messages you send to them and send you messages back.

The format for this is:

http://www.activitymonitor.net/YOUR_SURNAME/?username=THEIR_NAME

Each person also has a password so they are the only people that can retrieve messages destined for them.

Website user 1:

Name:

Password:

Their URL: http://www.activitymonitor.net/   /?username=

Website user 2:

Name:

Password:

Their URL: http://www.activitymonitor.net/   /?username=

Website user 3:

Name:

Password:

Their URL: http://www.activitymonitor.net/   /?username=

Website user 4:

Name:

Password:

Their URL: http://www.activitymonitor.net/   /?username=

Match Activity Monitor User Manual

Setting up a basic rule

The image below shows a basic rule setup. To do this click on the "Rules" tab at the top which will turn blue to indicate it is selected.

To change which rule you are editing click the number of the rule at the side. In this example we are editing rule 1.

Clicking the button icon at the top gives a list of buttons that you can select from. Select "My Buttons" and then click on the blue and red buttons to turn them green which means it is selected.

Click the people icon in the top right to get a list of people you can send these buttons to. In this case we select the User "Lucy". Now whenever we press one of those two buttons on the My Messages screen then Lucy will receive a notification of this. So you can use this to alert him/her quickly.
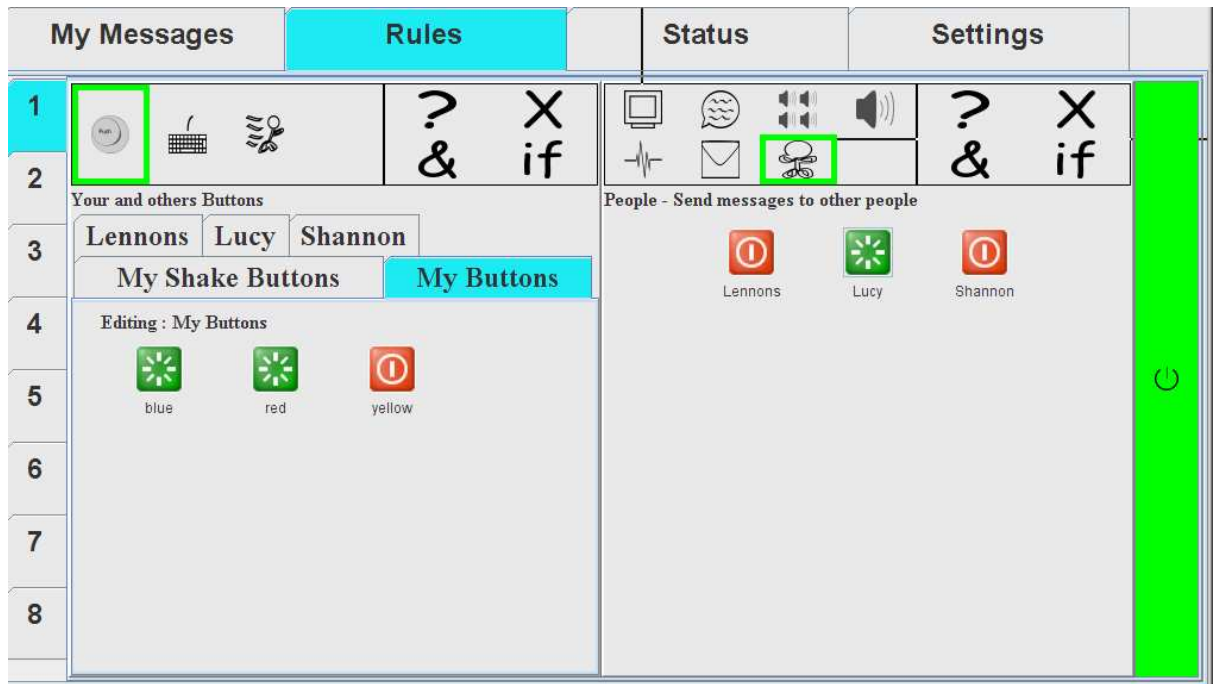
We might then want to set up another rule, so we can click on the 2 at the left to change the rule we are changing. This time we want to setup where all our text messages go. For example we could want that all text messages from anywhere appear on our list in My Messages and make a popup whenever it is in photo frame mode. Below is an example of how that might look having selected the keyboard icon and all the types of text message (Calendar, your own and text messages from others) and sent them to the GUIs.

What's it doing?

If we want to find out what it's doing right now we can click the Status tab at the top. This will allow us to see which Activities are being monitored (green lights next to them) and by clicking the Destinations we can see which places have activities being sent to them.

For example on the status below we can see that the Calendar, among other things is being used.

If we want to find out more than the fact that its simply being used we can click on it and it tells us where calendar messages will be sent. In this case to the GUIs due to the rule we just set up.

Sending Messages

We can send messages from the My Messages tab. Simply click the Send Typed Message button.

And type your message in on the on screen keyboard.

As we setup in the previous rule this gets delivered to the List of My Messages. So it appears right away.

However, if we look at the detail for My Text Messages on the Status panel we see that our own text messages are only being delivered straight back to ourself. Which isn't terribly useful.

So let's set up a third rule to send My Text Messages to everyone else as well.
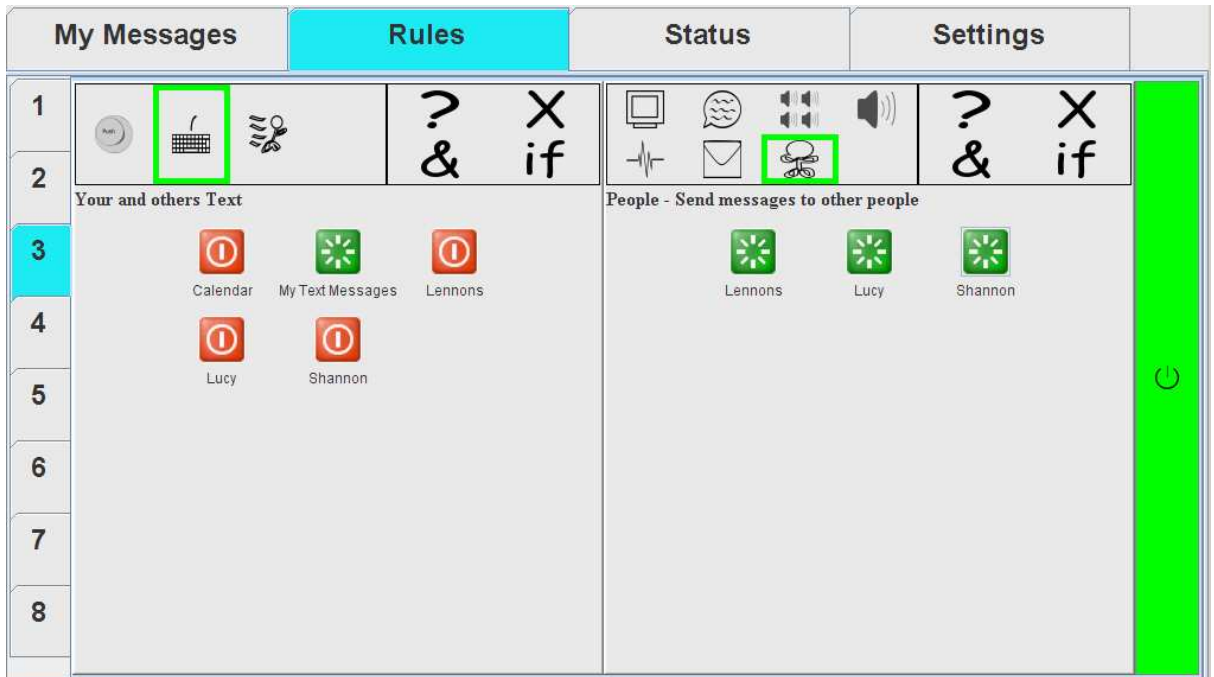
Now if we look at the details for My Text Messages we see that other people receive them as well which is more useful.

More complicated options

One of the more complex options is when instead of the buttons and text messages there are enough options under a menu that we need to split them into categories.

An example of this is the Movement option shown below.
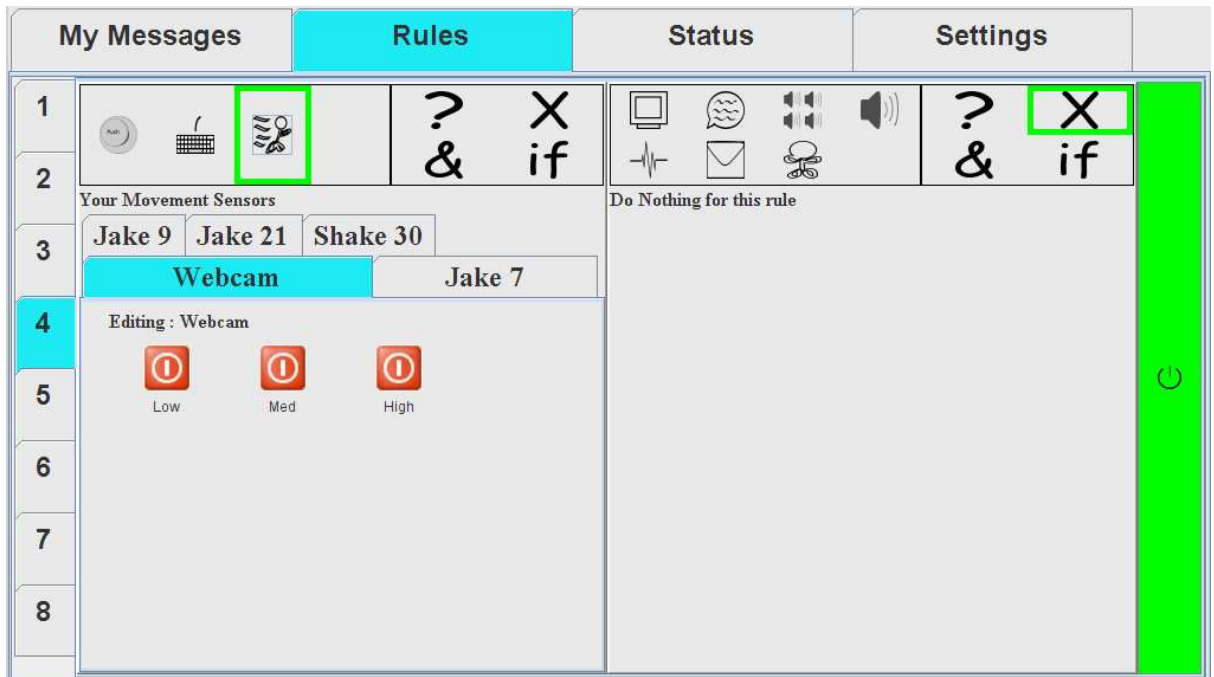
Here we see that we have a number of other small tabs inside it. Labelled "Jake 9, Jake 21, Shake 30, Jake 7 and Webcam". This just allows us to have more options available. So in this case we have the Webcam selected and we can choose Low, Medium or High sensitivity.
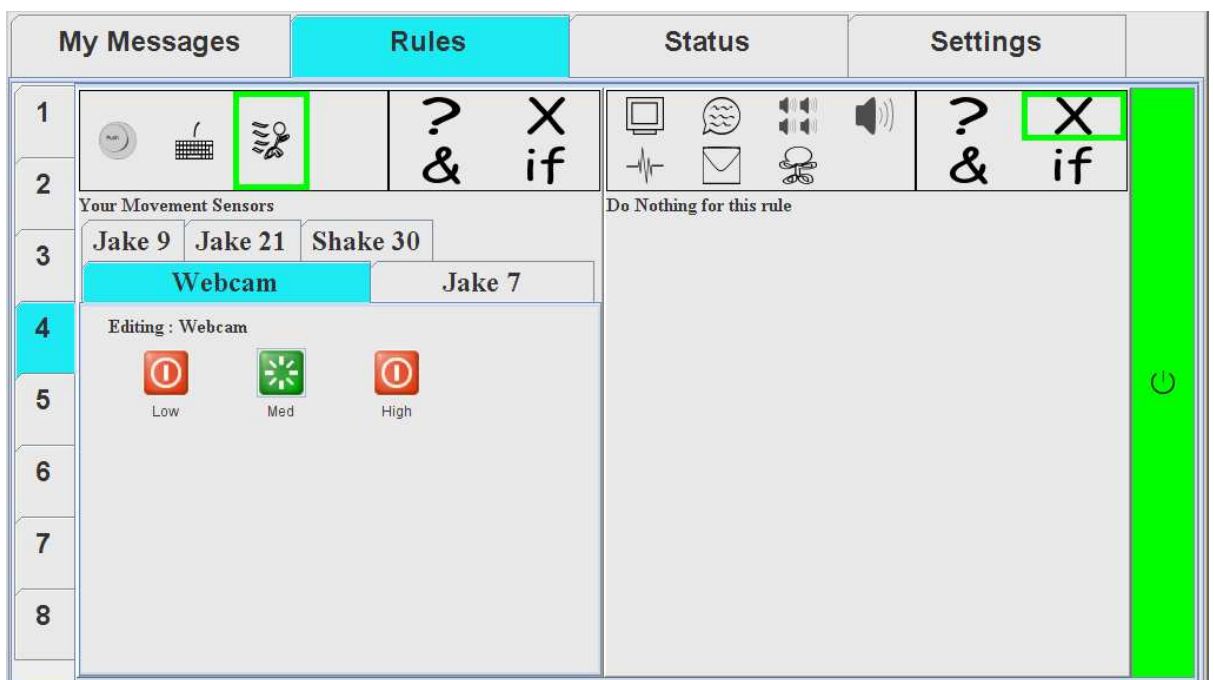
We have then selected Medium sensitivity for the webcam. This means that whenever the webcam detects movement it will send a message.

If you select Medium and then change your mind and change it to High later on then it will automatically unselect Medium for you. Since it doesn't make sense for it to be *both* medium and high sensitivity.

We can change the message sent for webcams under the settings tab as shown below.
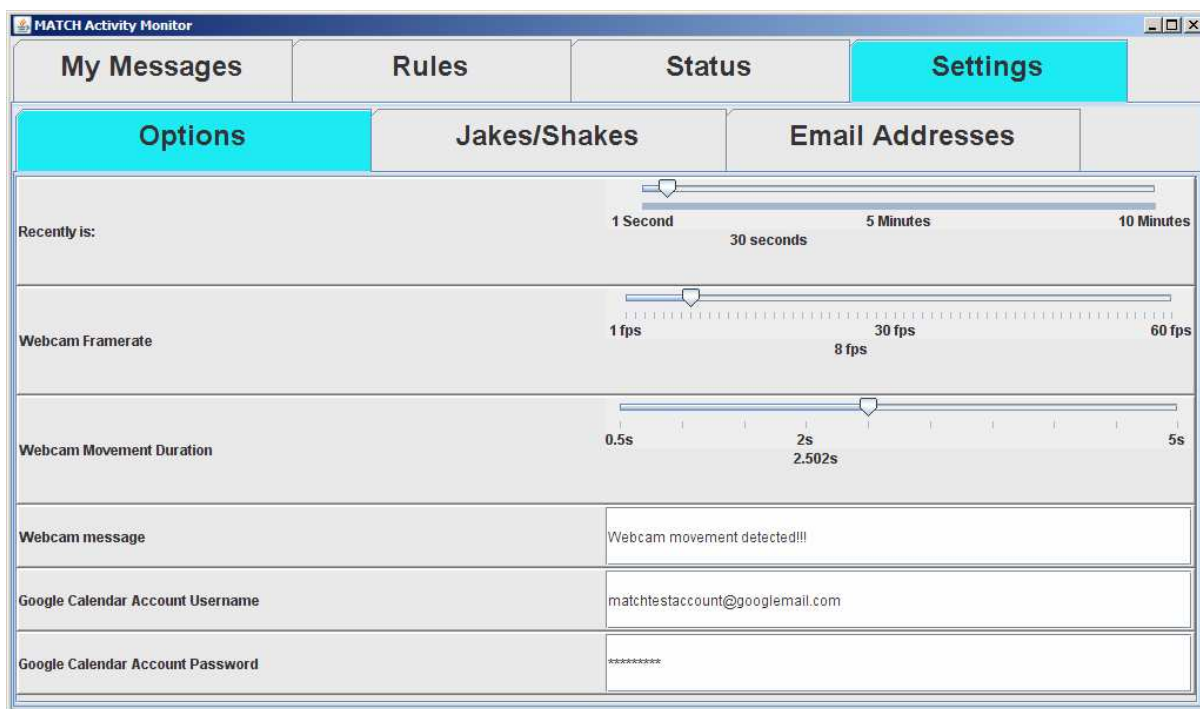
This also allows you to change how long ago "Recently" is.

You can also change the messages sent by the Jakes and Shakes under their options.
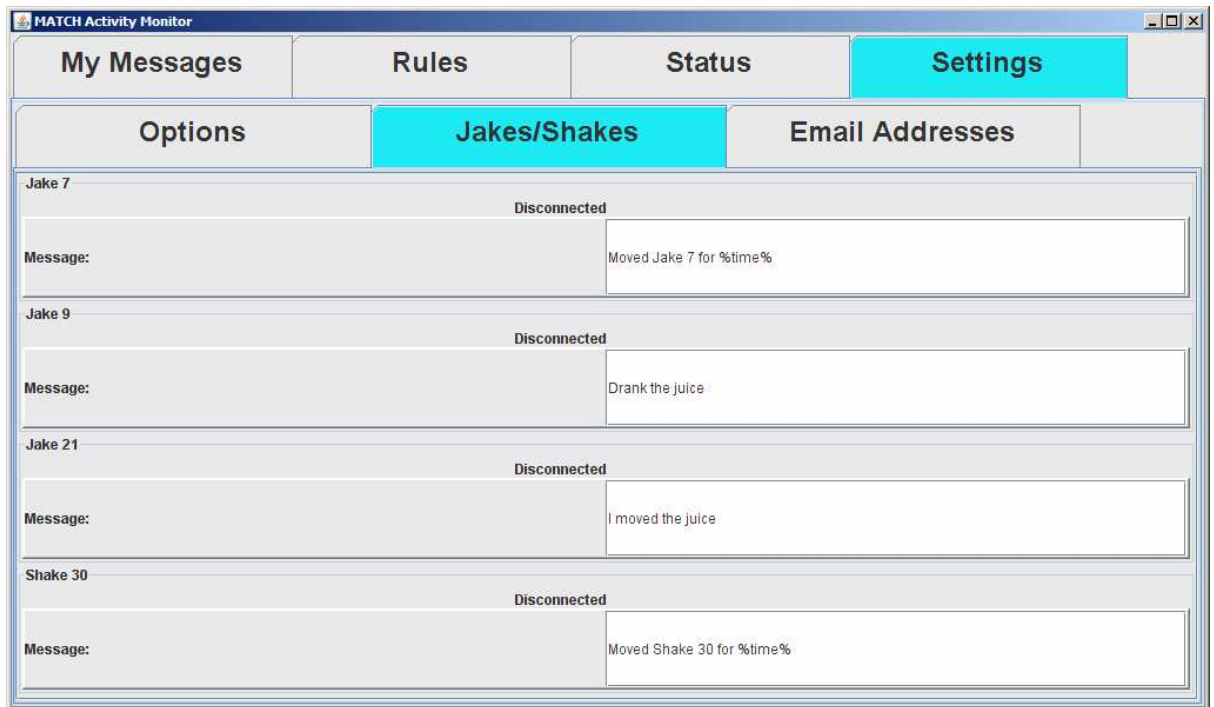
So you can specify a different message for each connected device.

AND Rules

When we send a Button press at the moment we don't get any feedback to say it's been sent. It just sends and then that's it. We might want to see a notification in Our Messages bit to say we've sent it successfully.

We can either set up another rule to do this, or we can go back to our first rule and change it to do both. So clicking on 1 again we select the & icon in the top right. Which allows us to pick two things instead of just one.
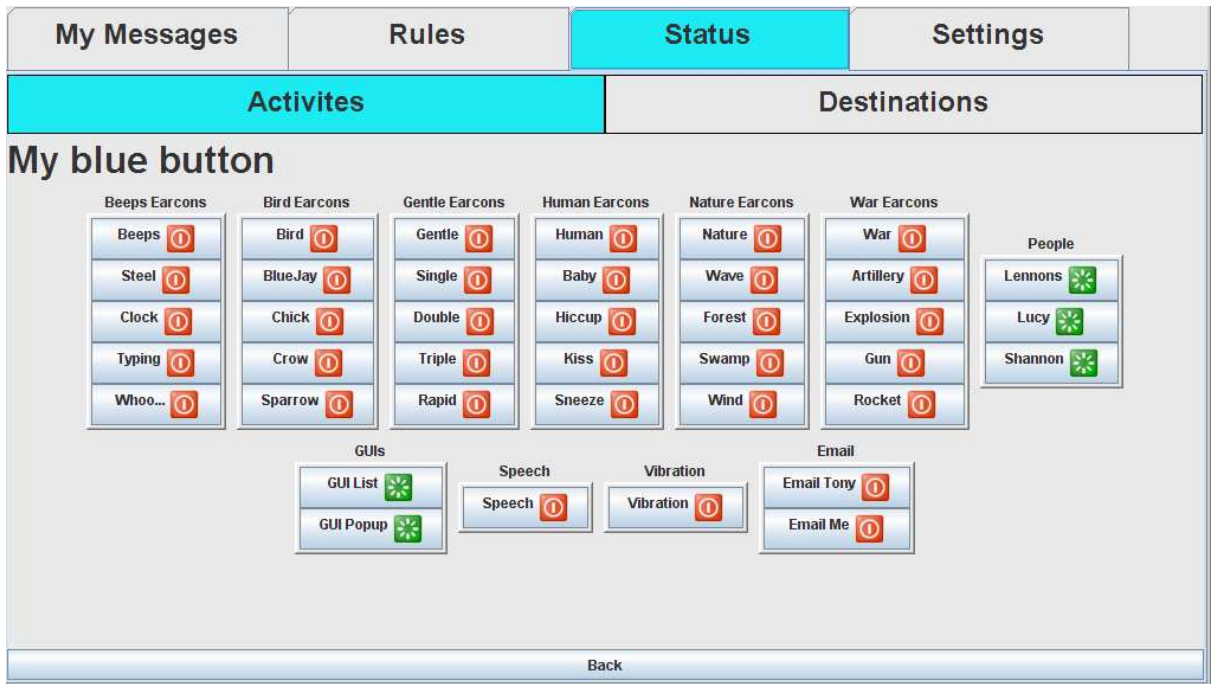
Under "Do this" we pick the first one (the GUI) so we can see it ourselves.
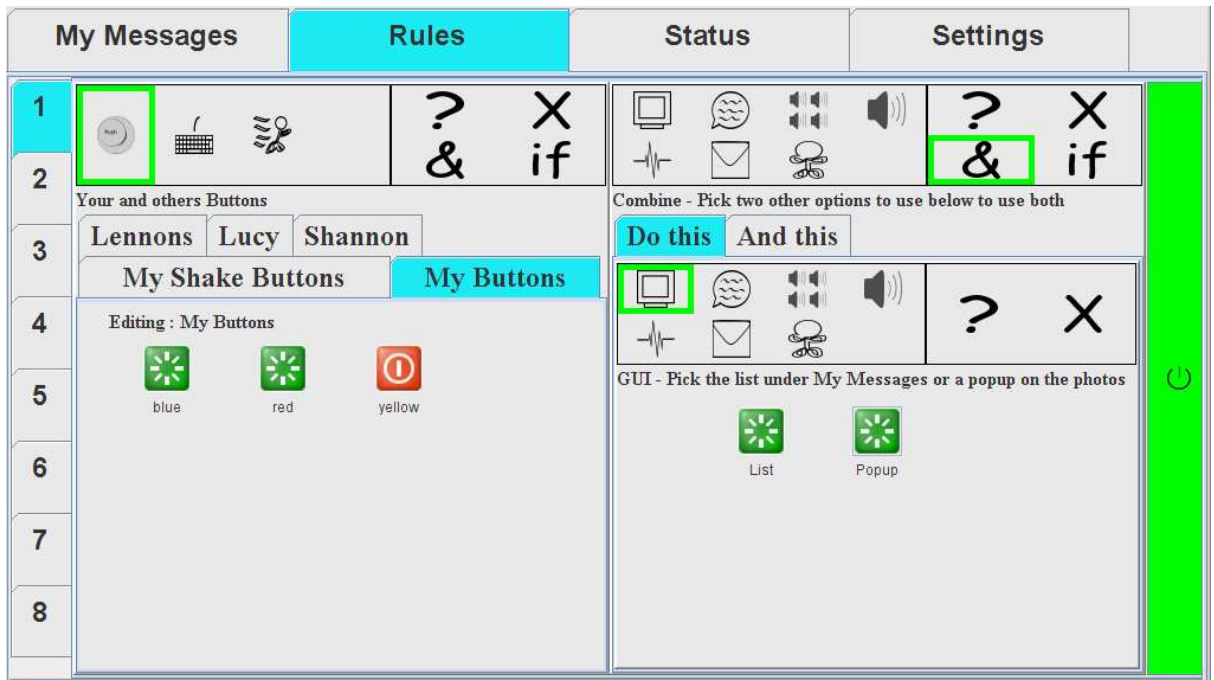
And under "And this" we can pick a second option. In this case the people option again so we can pick who those buttons get sent to.
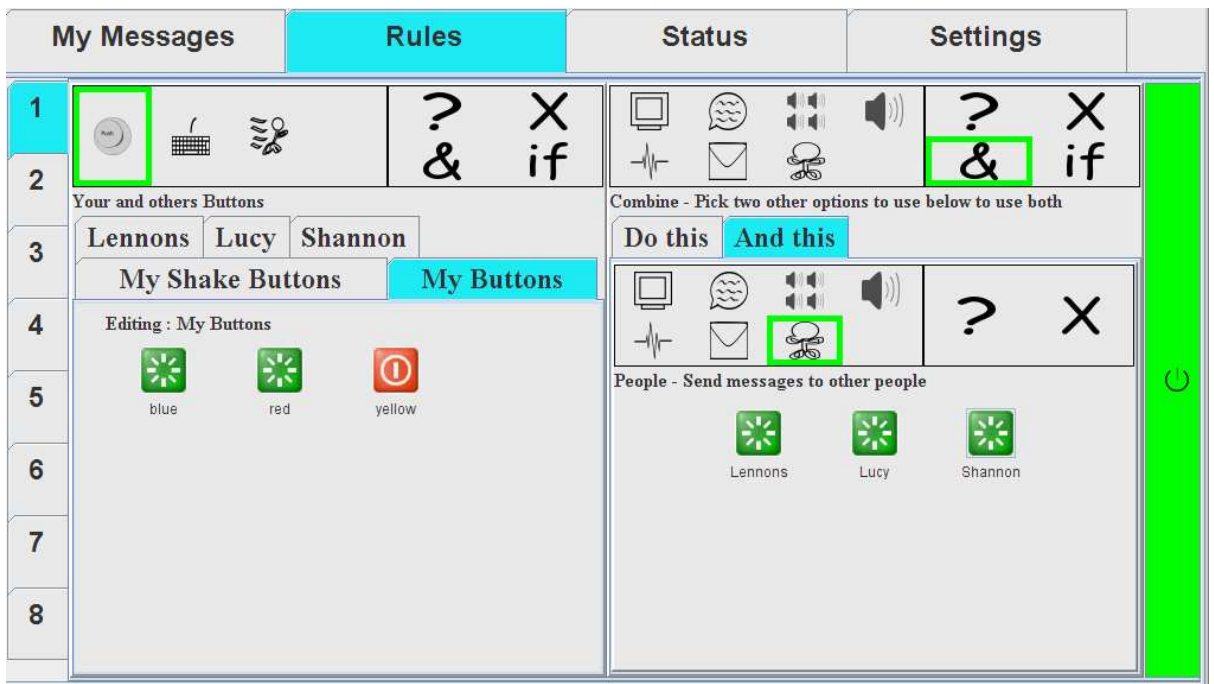
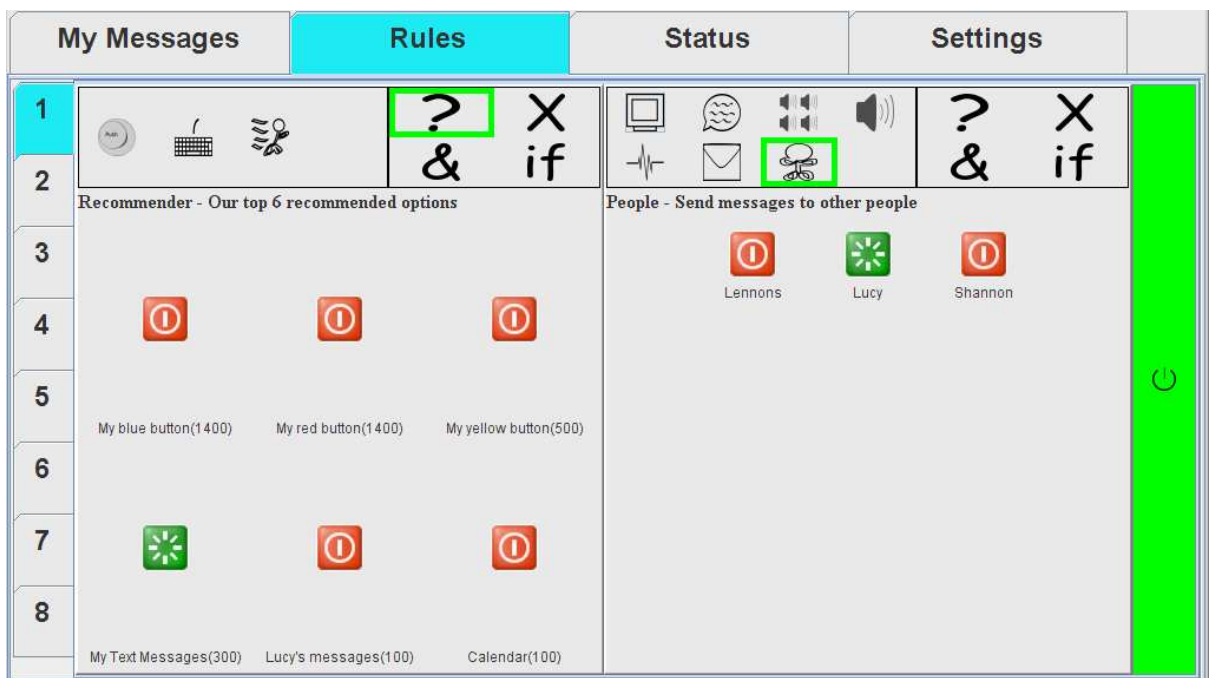If we look at the details for one of the buttons we can see that it is now going to our GUI and to People.

Recommender

There is also a "recommendation system" that you can use which you can access via the question mark symbol.

The recommender will recommend what it's opinion of the best six things you can use for one half of the rule are. You are then free to select as many or as few of these as you like.

In the above example the user has chosen Lucy as the destination and the recommender has suggested three buttons, text messages, Lucy's messages and Calendar.

Be aware the recommender can sometimes take a few moments to load and will get more accurate the longer you use the system.

Context Sensitive (if…)

It is possible to change what happens based on if you have moved recently or not.

To do this click the "if" icon. Select the "Sensor" tab to begin. First decide which sensor you want to count your "Moved recently" from.
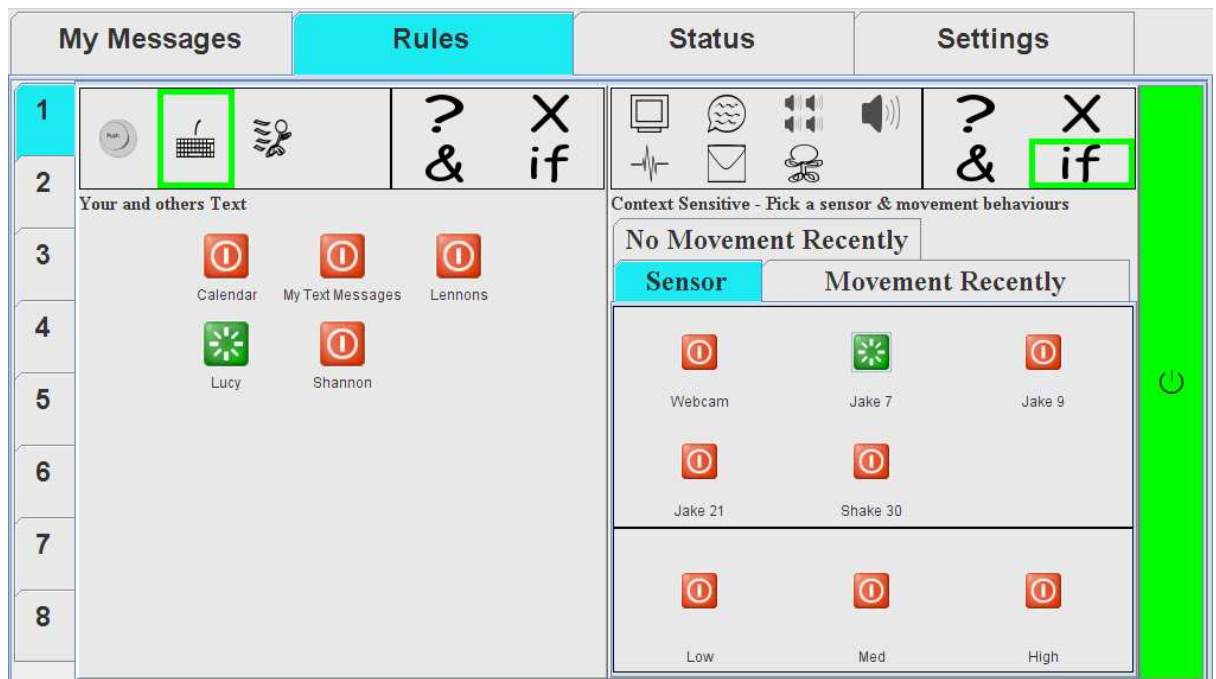
In this case we have chosen the "Jake 7" sensor. We also need to select the sensitivity. I.e. how much moving we need to do for it to count. Let's go with "High" and it will count just about everything.

Now we have decided our sensor we need to decide where to send it to in each case. So, if we click "No movement recently" we can then pick a rule within there like normal.

Here we have chosen "send an email to me". This is in case we are away. So whenever Lucy messages us and we haven't moved the Jake recently then it will go as an email.

Now to handle the Movement Recently case.
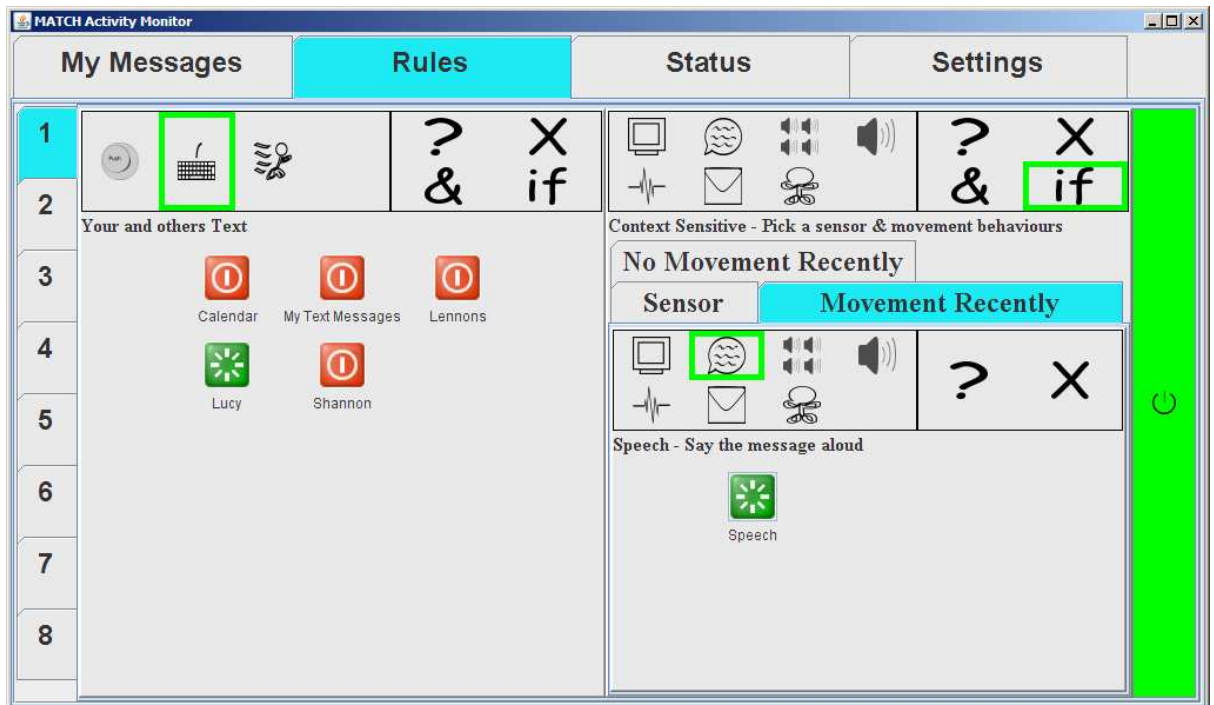
In this example we have said Speech. So if there has been movement recently and Lucy sends us a message then it will be spoken. We know we can just speak it because we've moved recently and we therefore know we must be about.

Summary

In this document we have restricted ourselves to only talking about Text Messages and Button messages and only used the GUI and other people as destinations. But there are many other things (Accelerometer movement, Calendar, Other Peoples Buttons) that you can get messages from and many other places (Speech, Earcon noises, Vibration, Email that you can send them to).

Experiment and play around.

Annotated cheat sheet

Movement

UMPC
screen

Speech

Groups of
sounds

Text
Messages

Individual
sounds

Buttons

Vibration

Email

Other
people

Switch
rule

Recommender

Clear

AND

Context
(if)

Turn a
rule on/off

Switch
between
categories
in an
option

Click to
select
devices in
a rule

My Messages   Rules   Status   Settings

Your and others Buttons

Lennons   Lucy   Shannon

My Shake Buttons   My Buttons

Editing : My Buttons

blue   red   yellow

People - Send messages to other people

Lennons   Lucy   Shannon