A Model-Based Approach to Supporting Configuration in Ubiquitous Systems

Tony McBryan[1], Phil Gray[1]

[1] Department of Computing Science,
University of Glasgow, Lilybank Gardens, Glasgow, G12 8QQ, UK
{mcbryan, pdg}@dcs.gla.ac.uk

Abstract. This paper presents an approach for representing, and providing computer support for, the configuration of interactive systems, particularly ubiquitous systems, that offers a flexible method for combining a wide range of configuration techniques. There are many existing techniques offering dynamic adaptation, ranging from fully automatic through context-sensitive to user-driven. We propose a model that unifies all of these techniques and offers a rich choice of ways of combining them, based on the concept of configuration possibilities, evaluation functions applicable to sets of these possibilities and approaches for parameterising the functions and combining the results. We present a concept demonstrator implementation of the model, designed for home care systems, and describe a set of use cases based on this prototype implementation that illustrate the power and flexibility of the approach.

Keywords: ubiquitous systems, dynamic configuration, model, evaluation

1 Introduction

Ubiquitous systems typically use large numbers of sensors to detect the state of the environment of use [1] and offer multiple different devices and methods of interacting with users [2]. The multiplicity and volatility of these contexts of use, including the presence or absence of devices and resources, especially when the users or devices are mobile, leads to a demand for systems that are capable of extensive and regular reconfiguration in regards to choice of interactive techniques and components. In addition, as the opportunities for reconfiguration grow, so does the likelihood that users will attempt to appropriate their systems to exploit this flexibility to provide new application functionality in new ways.

This situation has led to the development of software architectures and technologies that enable this dynamic reconfiguration to take place and also to the development of a variety of techniques for carrying out this configuration. The latter range from conventional preference settings through interactive configuration interfaces to autonomic context-sensitive systems that adjust the form of interaction to the current state of the user and setting; perhaps based on sophisticated policies or via matching to previous similar patterns of use. Each of these techniques is useful in certain circumstances and, indeed, combinations of the techniques are also possible.

From both a design and implementation point of view, it would be desirable to treat all of these techniques in a unified way, as variants of a single coherent model of configuration, so that they can be more easily compared, transformed, combined, refined and swopped. This paper presents such a model, based on the notions of configuration possibilities and evaluation functions over such possibilities. We shall argue that this model offers a rich design space for a range of configurations, making it easier to combine techniques and to develop new variants of existing ones.

In Section 2 we briefly review related work on the configuration of user interfaces to identify the techniques we wish to unify. Section 3 presents our model-based approach to configuration followed by Section 4 that describes a proof of concept based on a set of configuration examples in the home care domain, implemented using a software framework we have built. Section 5 offers our conclusions and an indication of future work.

2 Related Work

Many techniques for choosing an appropriate interaction technique or device have been developed in the context of ubiquitous systems design. In this section we summarise some of the most popular approaches with some exemplar implementations. This section is intended to discuss the use of the system from the perspective of a typical user and does not compare architectural features of particular approaches.

Thevenin and Coutaz [3] present the notion of plasticity that identifies equivalence of usability as the primary criterion for assessing interaction adaptation. Their implementation demonstrates automatic and semi-automatic generation of user interfaces exhibiting plasticity.

Manual configuration is frequently used to allow the user complete control over a configuration. Using a manual approach it is necessary for the user to specifically make a modification to the configuration when circumstances change. This configuration can be stored in a configuration file, possibly expressed in an appropriate specification language [4] but often commonly manipulated by an interactive editor such as Jigsaw [5] which uses a “jigsaw pieces” metaphor to enable a user to see the interconnection of devices and to manipulate them to meet changes in demand. Another similar approach is Speakeasy [6] that allows direct connections, as in Jigsaw, but also employs a task based approach where templates are “filled out” with the appropriate devices by the user.

Context sensitive systems are systems that choose the interaction techniques to use based on data gathered from the user’s environment – their context. Schmidt [7] describes a hierarchical model of context which includes the user model(s), social environment, task model, environmental conditions and physical infrastructure from which adaptations are derived.

Another approach is to define a “utility function” that automatically decides which interaction styles or devices should be used to communicate with the user. These utility functions may then make use of any contextual data gathered as part of the function. This is the approach taken by Sousa and Garlan [8] where a utility function

is used to express the combination of the user's preferences, the suppliers preferences and quality of service preferences. The task of making a choice is then an effort to maximise this utility function. This approach is also found in Supple [9] which performs user interface adaptation according to a utility rule based on pre-assigned weights for screen components.

Rule based reasoning can be used to select appropriate interaction techniques automatically based on rules or policies manually set by the user. In the work of Connelly and Khalil [10] this takes the form of policies for devices and interaction spaces being combined to determine the interaction methods that are allowed to be used. This approach is also a clear influence on the current work being undertaken by W3C Ubiquitous Web Applications [11] where content and presentation are selected based on selection rules based on the characteristics of the device(s) currently in use.

Another approach used by the Comet (Context of use Mouldable widgET) architecture [12] is to employ introspective components that publish quality of use guarantees for a set of contexts of use. 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 to make a decision on which component should be used. Each component must therefore be able to identify its own quality of use statistics in each of the contexts of use it is possible to appear in.

It is also possible to use "recommender" or collaborative filtering techniques to make the decision. A recommender algorithm may use a collection of preference or usage histories and compare them to similar information, either from the same user or from multiple users. This approach is used in the Domino system [13] to determine which components a user has access to using a history of frequently used components from other users.

A final approach to be considered is employed by the ISATINE framework [14] based on the USIXML mark up language. ISATINE is a multi-agent architecture that decomposes the adaptation of a user interface into steps that can be achieved by the user, the system and by other external stakeholders. The user can take control of the adaptation engine by explicitly selecting which adaptation rule to prefer from an adaptation rule pool in order to express the goal of the adaptation more explicitly but does not provide a mechanism to utilise multiple configuration techniques at run-time.

All of these techniques are useful in certain circumstances, but currently no system provides a unified method of offering them all, both separately and in combination. Our approach, described below, is intended to provide this unification.

3 Unified Model-Based Approach

Our approach to the configuration of interactive systems is to represent each of the techniques discussed in Section 2 within a unified model. This approach allows designers to provide many configuration techniques in parallel or in combination and are potentially modifiable at run-time and capable of being driven by user interaction.

3.1 An Application Context

Our work has been carried out as part of MATCH[1], a multi-university research project devoted to investigating infrastructure support for dynamically configurable, multimodal ubiquitous home care systems. For that reason, we illustrate our approach by the use of a running example taken from this domain. In this example 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 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.

3.2 A Unified Model of Configuration

The model we present 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.

To do this we introduce concept of a *configuration possibility* (hereafter, 'possibility', for short) which is an *encapsulated solution (consisting of interaction components, techniques and devices) that can offer interaction between a system task and a user*. A possibility includes any software components needed to perform data transformations related to the interaction as well as references to the components that will be responsible for rendering the interaction via physical devices.

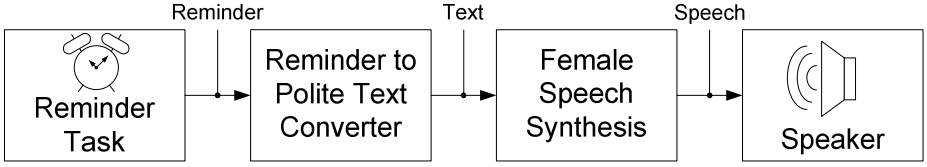

Fig. 1. A typical configuration possibility

Consider a medication reminder for Fred; one of the possibilities, as shown in Figure 1, might be to deliver the reminder via a speech synthesis system. The possibility would include the component representing the physical device (the speaker), the component representing the speech synthesis system (responsible for converting text to speech) and the component that converts a medication reminder into the appropriate textual alert.

To construct a set of possibilities it is possible to use a service discovery system that models relationships between components to construct a directed graph of the available components suitably configured. By identifying interactive components it is

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

possible to traverse the graph with the goal of constructing a set of possibilities that can be used with the application task.

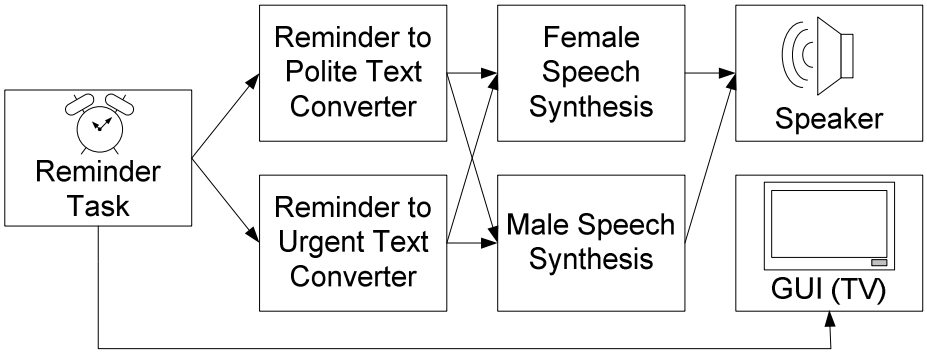

Fig. 2. A typical graph.

Figure 2 shows a typical, albeit simple, graph that may be constructed from the data in a service discovery system. In this graph we can deduce many different possibilities (such the speaker using polite text and a female voice); we have shown a speaker that requires the choice of two of the intermediate components as well as a GUI that does not require intermediate components. By starting from the reminder task as the root node we can perform as simple breadth first traversal to determine each possibility in the graph.

More complicated graphs including cycles will require a more robust traversal algorithm to determine every possibility. Some unanswered questions currently remain over the likelihood of graph explosion, and what impact this may have on performance, given unrestricted, large numbers of possibilities. This will be a subject of future research and is not addressed here; to date we have not experienced performance problems with graphs of moderate complexity (~70 nodes, ~120 edges).

Once the graph has been built and traversed to create a set of possibilities we can begin to analyse the appropriateness of each possibility. To do this we evaluate each possibility by using one, or many, evaluation functions.

The purpose of an evaluation function is to rank, filter or otherwise analyse these possibilities such that a configuration decision can be 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.

Figure 3 shows one possible result from the application of two evaluation functions (a ranking and an approval function) to some of the possibilities we could have 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.

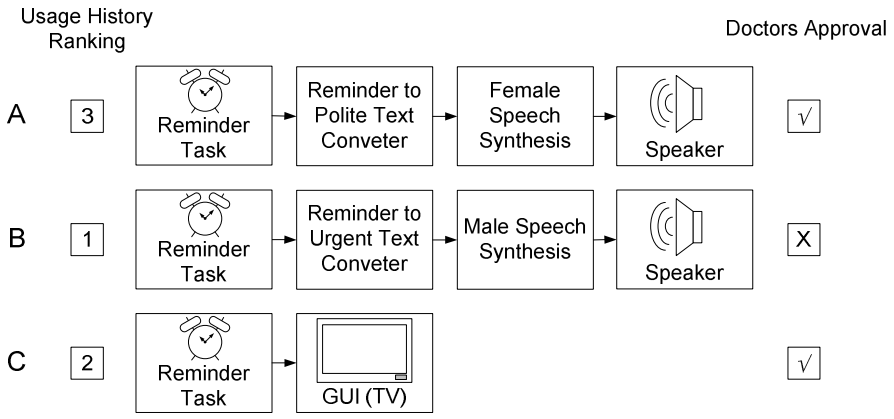

Fig. 3. Example results from the application of a ranking evaluation function and an approval evaluation function.

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

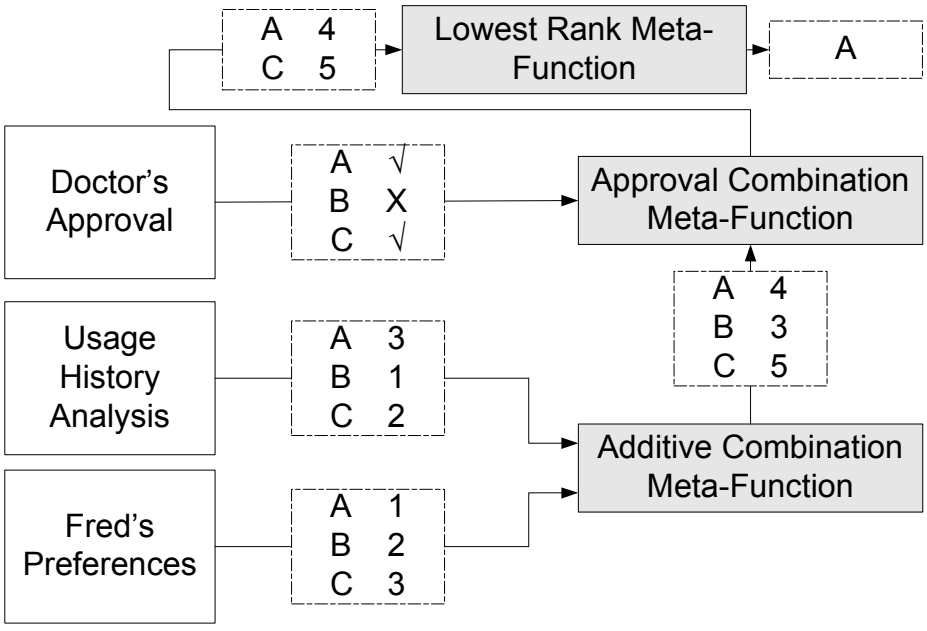

Fig. 4. Example results from the combination of three evaluation functions.

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 4 shows one possible method by which three evaluation functions (2 ranking and 1 approval) might be combined together in this approach to determine which possibility to use from the three available possibilities shown in Figure 3.

Two of the evaluation functions are implemented as ranking functions which "score" each of the possibilities. The individually ranked results of both ranking functions are first combined together using an additive meta-function before the results of this are combined with the results of the doctor's approval evaluation function. The result of this is that possibility 'A' was the possibility with the lowest combined rank that had also been approved and was therefore selected.

The meta-functions can be replaced or changed at will to provide different results, for example the choice of meta-function to combine the results of the two ranking functions could have instead been multiplicative in nature which may have had a different result.

A useful result of this is that the system has inbuilt support for multiple, conflicting stakeholders using the system. Each stakeholder 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 conflicts can be solved by changing the meta-evaluation function being used to combine the results.

The result of an evaluation function (or set of evaluation functions) should be the set of possibilities to use for interaction; as shown in Figure 4. In this case, a single technique has been selected, although functions might also 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 parameterisable 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.

3.3 Interactive Evaluation Functions

Evaluation functions can, and often must, be interactive components themselves. Users can (i) provide inputs prior to function creation or use (e.g., preference files read by a function), (ii) interact with an evaluation function directly as part of the evaluation process, (iii) indicate a changed opinion thus triggering a re-evaluation or (iv) interact implicitly, in which some evaluation functions gather usage information

or indications of the user's satisfaction over time to determine how to rank or filter possibilities.

Similarly, a meta-evaluation function can be interactive. In the example, in Figure 4, the "lowest rank" meta-evaluation function could be replaced with a function that presents the two remaining choices to the user along with the current rankings and asks them to choose which should be used.

The process of allowing for user interaction as a part of this process means that an evaluation process may need to be deferred until the user has responded. In this case a provisional decision may have to be made in the meantime to provide a service until the user has had sufficient time to complete their interaction.

Since we can combine approaches systematically, we can have a combination of automatic and manually-controlled evaluation function in use at the same time. We may also have policy-based evaluation functions mixed in – we may even have multiple different policy specification languages being used at any one time.

We envisage two primary modes of interaction: (i) one-off or sporadic interaction where the user specifies their needs and wants in advance and rarely changes them, and (ii) continuous interaction where the user frequently interacts with the system, or plans to interact with the system, to assist in the choice of suitable interaction techniques.

In addition, we believe that evaluation functions (and meta-evaluation functions) may be required to provide explanatory information or reviews on the current state of the system or on previous choices they have made so far; similar to the approach in the Crystal application framework [15]. This allows users to have an idea of the reasoning by which an interaction technique was chosen (why is the system behaving as it is?) or to be presented with the currently available choices and the ways in which the system can assess them (how might the system behave if changed?).

In summary this approach allows us to combine together automatic reasoning functions together with interactive functions within a unified model where conflicts between stakeholders can be represented explicitly.

3.4 Interaction Evolution

One of the aims of this approach is to support *interaction evolution*. The concept of evolution we use here is influenced by Dourish [16], MacLean [17] and Fickas [18]. Each of these authors identifies the ability to appropriate, tailor and evolve a system over time as a key feature of ubiquitous systems. We define interaction evolution as *multiple related instances of interaction configuration that have a directed goal to change some aspect of the system with respect to certain attributes of quality.* For example, an elderly user might develop a visual impairment (e.g., cataracts) that requires a reduction in dependency on conventional visual displays. Over time their visual capacity might deteriorate, perhaps resulting in the invalidation of the current configuration choice. Our approach enables us to build evaluation functions that operate over longer periods of time (sequences of choices), thus supporting such evolution by exploiting persistence.

4 Validation of Our Approach

In the remainder of this paper we will discuss an initial validation of our approach through example concept demonstrator applications, based on the scenario presented in Section 3.2 (see section 4.2 for more details).

4.1 The MATCH Software Framework

These demonstrators have been implemented in a software framework developed within the MATCH project. This section describes the architecture briefly; further details of the implementation of this framework are available in [19].

Within the framework architecture (Figure 5) sets of application tasks are controlled by a Task Manager component, responsible for starting, stopping and otherwise controlling tasks and their parameters.

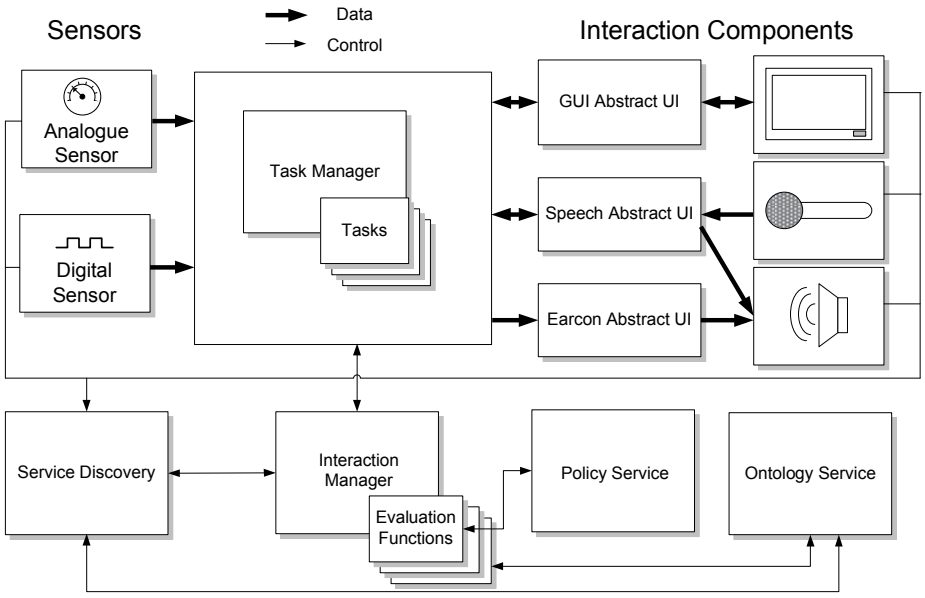

Fig. 5. MATCH Architecture

Components such as sensors and interaction components are provided as logical software "bundles" within the system which can be dynamically added and removed at runtime. Components are not limited to those which are locally accessible; for instance some components may be implemented as web services which are hosted remotely. Interaction components and tasks are registered with a service discovery system, supported by an Ontology Service [20], that can be used to hold high-level descriptions of components and tasks. Evaluation functions benefit from the

Ontology service which allows reasoning about classes of related components and their effects on the user based on the information held by the ontology service.

Communication between components and tasks is brokered by a publish/subscribe message handler.

The Interaction Manager subsystem is responsible for the implementation of the approach described in Section 3. When a task is started, it will request from the Interaction Manager any bindings to interaction components it requires. The Interaction Manager has a repository of assigned evaluation functions and will query the appropriate evaluation functions to determine the allocation. Evaluation functions can additionally notify the Interaction Manager that a change has occurred requiring re-evaluation, performed subject to meta-evaluation approval (to allow for deferral of re-evaluations).

Since some evaluation functions may be implemented as rules or policies we have provided a Policy Service [20] component which is capable of reasoning over sets of policies and is a service available to evaluation functions. Other services, such as alternative policy services, recommender services or usage history services could also be made available to evaluation functions to use.

In the rest of this section we present a number of use-case examples that have been built with this framework to demonstrate the basic suitability of our model for unifying automatic and interactive techniques for configuration. The implementations use a SHAKE [21] battery-powered multi-sensor pack equipped with accelerometer, gyroscope and magnetometer to detect movement traces. The interaction devices we use for this implementation are currently simulated versions of the actual devices mentioned in this section (e.g., TV and phone emulators) and the user interfaces to the evaluation functions remain primitive.

4.2 Scenario for the Demonstrator Applications

Recall that Shirley has worsening arthritis restricting her mobility. Fred wants to be informed about Shirley's activity levels so that he does not worry. Fred is interested in seeing this data on his mobile phone both at home and away. He does not need to be notified about the status if he is currently in the room with Shirley since he can observe for himself. The monitoring data is of interest to external agencies such as Shirley's doctor who would like to be kept apprised of changes in Shirley's condition.

To this end Shirley wears a wireless accelerometer that captures her movement in real time and delivers it to the MATCH framework as a sensor stream. A task exists in the framework that interprets the raw sensor data and generates notifications when there has been little movement or unusual movement patterns.

4.3 Example 1 – Utility Function, Multiple Resolutions

We can imagine that Shirley's doctor has prepared an evaluation function which selects a "default" hardcoded configuration. This evaluation function is designed to advise both himself and Fred of Shirley's condition on an ongoing basis. This default

evaluation function is a utility function designed to maximise benefit by using pre-selected interaction components.

Utility functions are the simplest type of evaluation function to implement as they can be completely self-contained and use extremely simple logic to perform their task.

As discussed in Section 3 an evaluation function has as input a set of possibilities available and returns as an output the set of possibilities to select.

In this case the set of available possibilities may include:

- SMS to the doctor's phone (perhaps provided for emergency conditions or for another task)
- HTTP post submission to a shared monitoring screen at the doctors surgery
- A television in the living room
- A loudspeaker which is audible throughout the house
- A monitoring application on Fred's mobile phone

The utility evaluation function is hardcoded to select the HTTP post submission as well as the audible loudspeaker and will simply return both of these possibilities which are both started, discarding all other possibilities.

4.4 Example 2 – Manual Configuration

Since the previous approach was entirely hardcoded it does not specifically address Fred and Shirley's needs for the monitoring application; it does not deliver the required information to Fred's phone and 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. To implement a manual choice in the form of an evaluation it is only necessary to create an approval style evaluation function that knows the user's choice and only approves the appropriate possibility.

In this scenario Shirley has created a connection via the HTTP based surgery monitor and manually adds and removes connections to Fred's phone and to the television in the living room depending on whether or not Fred is home.

4.5 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 what is actually required is to use the preferences evaluation function.

Fred selects a set of preferences (Phone > TV > Loudspeaker) and changes the monitoring task to use the preferences evaluation function with his set of preferences.

The evaluation function will take the set of available possibilities and return a single possibility of the highest preference, i.e. if the phone is available then the phone possibility will be used, otherwise the television and finally the loudspeaker.

Since the system only considers available possibilities Fred starts turning his phone off when he's in the house so that it is marked as unavailable and cannot be selected. This causes his second preference, the television, to be used.

4.6 Example 4 – Combining Evaluation Functions

Previously the preferences were configured only for Fred's usage and ignored the needs of the doctor who needed to monitor Shirley's condition over a period of time.

Thus it is necessary to combine the doctor's needs with Fred's preferences. To do this, the simplest approach is to have two evaluation functions – one for the doctor's needs and one for Fred's. One evaluation function selects the doctor's surgery monitoring application, if available, and otherwise the SMS function, the other duplicates the preferences in the previous example.

These can both be implemented as two instances of the same basic preferences evaluation function but with different sets of preferences.

In order to combine these evaluation functions we can use a meta-evaluation function (election system) to the task which operates over a selection of sub-evaluation functions. When the meta-function is queried it simply queries each sub-function in turn and returns as its result the union set of the results from each sub-function. In this case it would return the set of the result of the doctor's preferences (the surgery monitoring application) and Fred's preferences (the phone or television depending on availability).

We could extend this to add an evaluation function for Shirley which may provide an "anti preference", i.e. devices she doesn't ever want used which may have higher precedence than the meta-evaluation function discussed here.

Other tactics of combining evaluation functions could be formed by providing alternate meta-evaluation functions (i.e. the intersection or union of the results of multiple approval functions).

4.7 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 interfaces 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.

This can be extended further by simply turning the context sensitive function into a switch between two sub-evaluation functions – your preferences in one situation vs. your preferences in another situation. This can be further extended to create logic trees of evaluation functions which control the sub-evaluation functions to be used.

It is also 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 doctors phone) rather than using the passive monitoring provided by the surgery.

5 Conclusions

In this paper we have presented a model-based approach to supporting configuration. This approach allows for the combination of multiple techniques ranging from fully automatic to fully interactive approaches for configuration and including various intermediate combinations.

The approach described here expressed composition and function without using a specific specification or description language but instead supports the combination of multiple disparate languages (for example; Java, ACCENT [22], MATLAB) within a single configuration if so desired. This approach is intended to be realised as a tool-supported configuration system where evaluation functions can be combined together and specified by the stakeholders. However, it may prove useful to express configurations in the model via a custom language.

Our initial examples, described above, only involve the selection and configuration of output components. We are now extending our use cases to support the selection, combination and configuration of components involving both input and output. We are working on more sophisticated interactive meta-evaluation functions, including their user interfaces, intended for typical users of a home care system. We are also working on applying techniques from voting systems to the model by viewing evaluation functions as voters in an election and meta-evaluation functions as the election systems themselves.

In the longer term, we believe that this approach is more broadly applicable than we have described here, including the selection and configuration of application tasks and sensors and involving multiple stakeholders with conflicting requirements. This will be the focus of further research.

6 Acknowledgements

This research was carried out within the MATCH (Mobilising Advanced Technologies for Care at Home) Project funded by Scottish Funding Council (grant HR04016). We wish to thank our MATCH colleagues for their contribution to the ideas presented here and for their work in developing the MATCH software framework.

7 References

1. Dey, A.K. and Mankoff, J.: Designing mediation for context-aware applications. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(1):53--80, (2005)
2. Oviatt, S.: Ten myths of multimodal interaction. Communications of the ACM, 42(11):74-81, (1999)
3. Thevenin, D. and Coutaz, J.: Plasticity of User Interfaces: Framework and Research Agenda. *Proceedings of Interact, 99*:110-117, (1999)

4. Magee, J., Dulay, N., Eisenbach, S. and Kramer, J.: Specifying Distributed Software Architectures. *Proceedings of the 5th European Software Engineering Conference*:137-153, (1995)
5. Humble, J., Crabtree, A., Hemmings, T., Åkesson, K.P., Koleva, B., Rodden, T. and Hansson, P.: Playing with the Bits-User-configuration of Ubiquitous Domestic Environments. *Proceedings of the Fifth Annual Conference on Ubiquitous Computing, UbiComp2003*, Seattle, Washington, USA:12-15, (2003)
6. Edwards, W.K., Newman, M.W., Sedivy, J., Smith, T. and Izadi, S.: Challenge: Recombinant Computing and the Speakeasy Approach. *In Proc. MOBICOM'02 - The 8th Annual International Conference on Mobile Computing*. pp. 279--286 (2002)
7. Schmidt, A., Beigl, M. and Gellersen, H.W.: There is more to context than location. *Computers & Graphics*, 23(6):893-901, (1999)
8. Sousa, J.P. and Garlan, D.: Improving User-Awareness by Factoring it Out of Applications. *Proc System Support for Ubiquitous Computing Workshop (UbiSys)*, (2003)
9. Gajos, K., Christianson, D., Hoffmann, R., Shaked, T., Henning, K., Long, J.J. and Weld, D.S.: Fast and robust interface generation for ubiquitous applications. *Proceedings of Ubicomp'05*, (2005)
10. Connelly, K. and Khalil, A.: Towards Automatic Device Configuration in Smart Environments. *Proceedings of UbiSys Workshop*, (2003)
11. W3C Ubiquitous Web Applications, Content Selection for Device Independence (DISelect) 1.0, http://www.w3.org/TR/2007/CR-cselection-20070725/
12. Calvary, G., Coutaz, J., Daassi, O., Balme, L. and Demeure, A.: Towards a new generation of widgets for supporting software plasticity: the "comet". *Preproceedings of EHCI/DSV-IS*, 4:41--60, (2004)
13. Bell, M., Hall, M., Chalmers, M., Gray, P. and Brown, B.: Domino: Exploring Mobile Collaborative Software Adaptation. *LNCS*, (2006)
14. Jaquero, V.L., Vanderdonckt, J., Montero, F. and Gonzalez, P.: Towards an Extended Model of User Interface Adaptation: the ISATINE framework. *In Proc. Engineering Interactive Systems 2007* (2007)
15. Myers, B.A., Weitzman, D., Ko, A.J. and Chau, D.H.: Answering Why and Why Not Questions in User Interfaces. *In Proc. ACM Conference on Human Factors in Computing Systems.* pp. 397-406, Montreal, Canada (2006)
16. Dourish, P.: Developing a Reflective Model of Collaborative Systems. *ACM Transactions on Computer-Human Interaction*, 2(1):40--63, (1995)
17. MacLean, A., Carter, K., Lovstrand, L. and Moran, T.: User-tailorable systems: pressing the issues with buttons. *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*:175-182, (1990)
18. Fickas, S.: Clinical Requirements Engineering. In Proc. ICSE 2005. *Proceedings of the 27th international conference on Software engineering*. pp. 140--147. ACM (2005)
19. Gray, P., McBryan, T., Martin, C., Gil, N., Wolters, M., Mayo, N., Turner, K., Docherty, L., Wang, F. and Kolberg, M.: A Scalable Home Care System Infrastructure Supporting Domiciliary Care. University of Stirling, Technical Report CSM-173 (2007)
20. Wang, F., Docherty, L.S., Turner, K.J., Kolberg, M. and Magill, E.H.: Services and Policies for Care at Home. *In Proc. International Conference on Pervasive Computing Technologies for Healthcare*. pp. 7.1-7.10 (2006)
21. Williamson, J., Murray-Smith, R. and Hughes, S.: Shoogle: excitatory multimodal interaction on mobile devices. *In Proc. SIGCHI conference on Human factors in computing systems* (2007)
22. Turner, K.J., Reiff-Marganiec, S., Blair, L., Pang, J., Gray, T., Perry, P. and Ireland, J.: Policy Support for Call Control. Computer Standards and Interfaces, 28(6):635-649, (2006)