A Generic Approach to the Evolution of Interaction in Ubiquitous and Context-Aware Systems

Tony McBryan and Phil Gray

Department of Computing Science, University of Glasgow, Glasgow, G12 8QQ

Abstract. This paper presents a model-based approach to the problem of evolutionary adaptation of ubiquitous and context aware systems where it is difficult or impossible to predict in advance the resources available, the criteria for judging the success of the change and the degree to which human judgement must be involved in evaluation process. The model is introduced via a simple example based around the adaptation of web content to different output device configurations. The relationship of the model to software architectures is illustrated via an example implementation in a homecare software framework. We argue that the model offers greater flexibility and control than other current approaches.

1 Introduction

The work presented in this paper concerns the adaptation of interactive systems where it is difficult or impossible to predict in advance (i) the resources available, (ii) the criteria for judging the success of the change and (iii) the degree to which human judgment must be involved in the evaluation process.

This is particularly of relevance in ubiquitous and context aware systems which must respond to a wide range of factors. Additional difficulties are experienced when new devices or evaluation techniques are added and it is necessary to evolve the system using these new techniques. These challenges have been recognised by Dourish[1] and MacLean[2] who argue that continual adaptation of interactive systems should be regarded as the norm rather than the exception.

Work in this area is aimed at addressing evolving systems such as ubiquitous systems within a home environment. The need for constant reconfiguration of devices within a home environment has been discussed by O'Brien and Rodden[3, 4] who recognise that items within the home as well as the home itself are 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.

The work presented here is a model-based approach to the dynamic configuration of interactive systems, based on the concept of generic and specialisable evaluation functions. These evaluation functions are responsible for analysing the options available for evolution to determine the optimal choice. Evaluation

functions can support varying modes of use and can be combined to allow novel support for configuration of interactive systems.

It is claimed that such an approach allows for systems that support a superset of currently available techniques used for the configuration of interactive systems. Additionally, the approach provides greater flexibility 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 adaptation of an interaction can be seen as a sequence of opportunities to change or evolve the interaction from its current state into a better state, in terms of some relevant criteria (e.g. user preference, efficiency).

This idea allows for flexibility in two key areas - the choice of evaluation method and the mode of use. The evaluation functions used for ranking the choice of possibilities can be replaced or combined to allow the user and/or system to prioritise the attributes of interactions they care about the most. The mode of use of multiple parts of the system can be customised to allow for different levels of manual or automatic control over the choice of interaction.

In the next section, we will illustrate this process with a simple example before discussion a generalised version in Section 3. We then consider the implications of our approach for the software architecture of systems that it. Finally we discuss related work and conclude with some observations on the current status and planned development of our work.

2 Example

For the purposes of this example we will consider a subset of the Arch model[5]. The Arch model provides a top down functional view of the components involved in an interaction in terms of 5 functional elements; the functional core, the functional core adaptor, the dialogue controller, abstract user interaction and concrete user interaction.

The functional core represents the domain dependant information involved in the interaction which communicates with the dialogue controller representing the task oriented goal of the interaction using the functional core adaptor which is provided to facilitate mediation between these two components. The Arch model includes notions of abstract and concrete user interaction components; the abstract components represent toolkit-independent interaction techniques which are implemented by or use a concrete component which actually performs the interaction.

To maintain simplicity in this example we will consider only the mapping between the dialogue controller, abstract user interaction components and concrete user interaction components although the methods described here can be extended to include the entire Arch model. We specifically deal with the choice of abstract and concrete user interaction elements as shown in Figure 1.

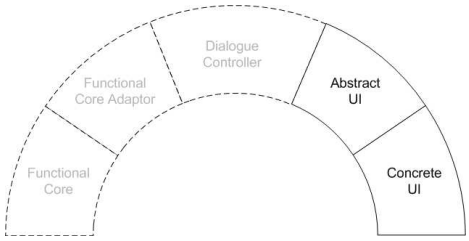

Fig. 1. Arch model with Relevant Components Highlighted

2.1 Possibilities

The Arch model may have many possible instantiations which could be used and these will be referred to as the possibilities for interaction. Here we consider all possible instantiations which are syntactically correct - that is combinations of components which are capable of interoperation, even if the choice of assembly forming the possibility is suboptimal. We will discuss the process to generate these possibilities later in Section 2.4.

A practical example of abstract and concrete UI's can be obtained from MacKay, Watters and Duffy[6]. MacKay et al. discovered that using three different layouts for viewing webpage's on a portable device resulted in very different advantages in different contexts. The authors investigated three different layout methodologies; (i) gateway - a zoomed out overview of the page, (ii) linear - a document transformed to remove horizontal scrolling and (iii) direct - offering a viewport onto the document as rendered on a desktop browser.

Fig. 2. Transformed Browser Windows from [6]

MacKay et al. found that linear layouts are advantageous when the user is unfamiliar with the page or when processing power on the device displaying the page is limited, but that gateway and direct layouts were preferred for familiar sites where the user knew the content already. These three layout techniques

could be implemented as three particular Concrete User Interfaces that can be used with the "display page" abstract user interface.

2.2 Evaluation Functions

It is then possible to apply different evaluation techniques in order to analyse the set of possibilities in respect to a variety of different criteria. For the purposes of this example we can use a simple ranking method where each evaluation function returns an integer in the range 0-10 as a ranking where the integer value 0 indicates an extremely poor result for this metric and the integer value 10 indicates an exceptional result. The evaluation functions used in that case are shown in Figure 3. This is a very simple structure and would not be suitable in many situations due to lack of precision and range and is intended only for clarity within this example; we discuss more generalised ranking features within Section 3.

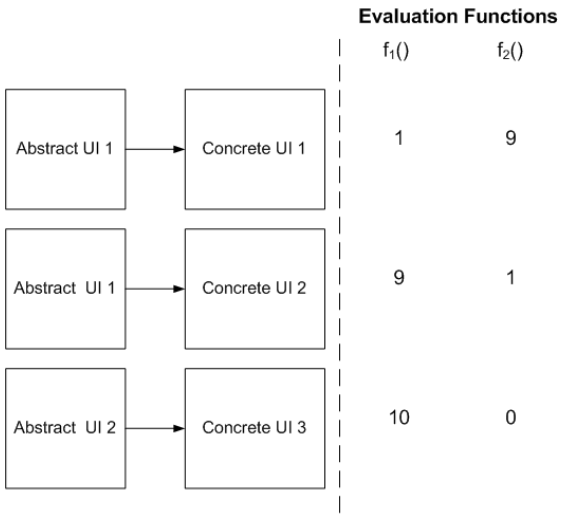

Fig. 3. Example Evaluation Function Results

To ground this more fully we refer again to the three mobile browser layout algorithms presented by MacKay et al. The evaluation functions used in that circumstance might be "predicted navigation performance" (as predicted by previous experiments or GOMS analysis) or "user preference" to rank each layout technique in terms of these measures.

2.3 Making a choice

Once evaluation functions have been applied to the set of possibilities we need to be able to combine these results to obtain a final ranking of possibilities. This final ranking allows us to choose the "best" interaction components to use; as judged by the evaluation functions.

A simple approach for our rankings is to simply weight the results by an individual factor for each evaluation function and take the minimum of each weighted result as our final ranking as demonstrated in Figure 4. This particular evaluation function combination technique has the property of finding the interaction which has the best worst-case performance and suppressing possibilities which receive low ranking evaluations. This combination method is unlikely to be used in a deployed architecture but it has the property of simplicity that is important in this example.

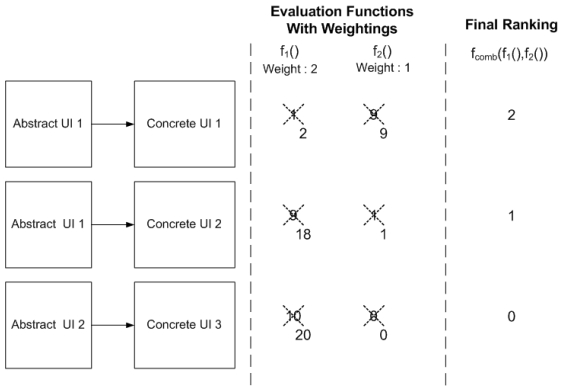

Fig. 4. Example Ranking

To continue the example from MacKay et al. we might have a circumstance that the "user preference" evaluation function has been highly weighted to increase its significance compared the "navigational performance" evaluation function so that two of the three options (direct and gateway) are equally preferred by the user and that greater navigational performance in the gateway example causes it to be chosen as the concrete user interface to use.

2.4 Graphing Possibilities

Section 2.1 introduced the idea of sets of possibilities which we can evaluate to determine suitability for use in an interaction. In this section we will elaborate on a technique of obtaining this set of possibilities.

The principal concept in this stage of the interaction choice is the construction of a directed graph representing the dependencies, or their ability to interoperate, of the set of available components. This can be constructed by starting with the dialogue controller as the root node and adding a new node for every abstract or concrete user interaction component. The a directed edge is created between the dialogue controller and every abstract interaction component that it supports, we then repeat this by creating directed edges between the abstract user interaction components and the implementing concrete components.

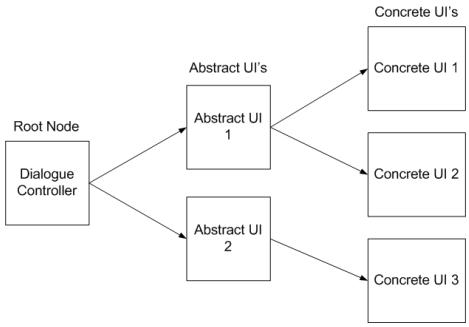

Fig. 5. Example Graph

In Figure 5 a tree, rooted at the dialogue controller, has been created where edges represent compatibilities between components. It is then possible to generate a set of possibilities analytically by performing a traversal across the graph starting from the dialogue controller; this can be performed using algorithms for single source all-pairs traversals.

2.5 Overview

The previous sections presented the individual steps involved to generate a sequence of possibilities, evaluate the possibilities and derive a choice of interaction components. These steps are shown in Figure 6 in order. This technique will now be generalised in the following section.

3 Generalisation

This technique can be generalised in several ways to support multiple modes of use (manual, automatic, semi-automatic) while at the same time allowing runtime adaptation of the criteria for decision making of interaction choices.

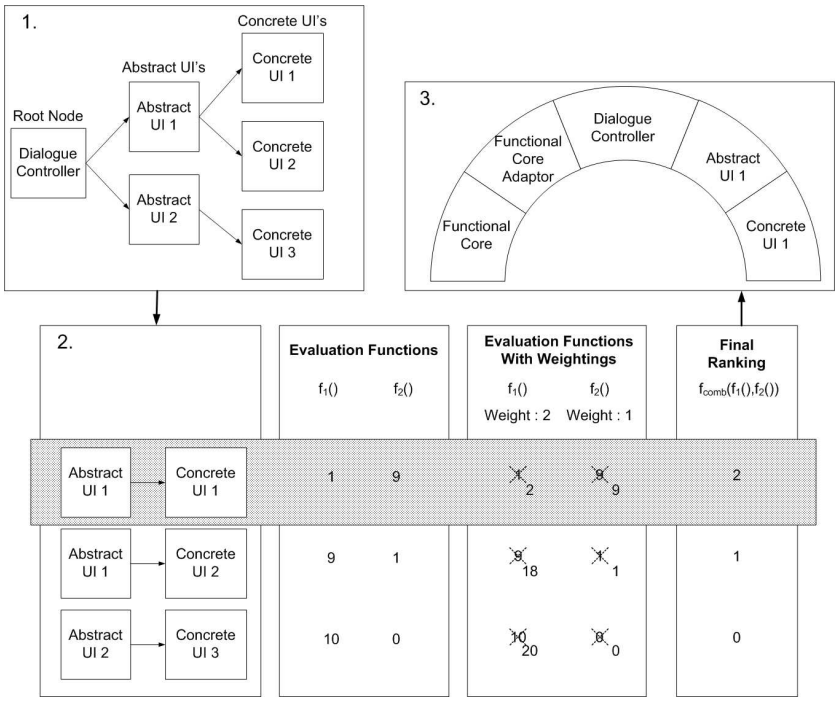

Fig. 6. Overview

3.1 Representing Possibilities

The aim of possibilities within this model is to represent the options that are available to be selected from within a system. Within an interactive system this would be the choices of interaction components to use to communicate with the user. Each possibility is a particular option for interaction and each of the possibilities can be reasoned about by evaluation functions.

Possibilities can be derived in various modes of use. Within the previous example possibilities were generated automatically from a graph of dependencies but these possibilities could also have come from direct user data entry or analysis of historic usage trends. Direct user entry of possibilities, either at design time or at runtime, could be used to restrict the selection of interaction component to only those specifically allowed by the designer. Likewise, analysis of previous usage of components could produce a set of highly used possibilities which could be used in preference to, or in addition to, an automatic deduction of possibilities from a graph.

3.2 Evaluation Functions

A set of evaluation functions can be used to represent different criteria that could be applied to each option for evolution. The aim is that each evaluation function is capable of ranking possibilities according to their own evaluation function that determines suitability. Evaluation functions essentially sort a list of possibilities according to their criteria for a good interaction component.

There would be a great selection of possible evaluation functions that could be available for use when evaluating a possibility; these would include functions that evaluated based on the intended user's location, user preferences, required computational resources, the user's physical abilities to interact with different interaction techniques and the discreetness of a particular interaction device. A breakdown of possible factors that could influence the choice of interaction component within a ubiquitous system is provided by Schmidt et al.[7] and each of these are candidates for evaluation functions.

In addition to these analytical evaluations we can use machine learning or recommendation techniques to analyse possibilities based on previous history of interactions. That is the evaluation function may be usage based and would compare the set of possibilities against previous history and rank possibilities which had previously found to be good matches for the user with higher relative ratings than those which had not.

It would also be possible to have evaluation functions which interact with the user in order to perform their ranking. A simple example of this would be functions which gather user preferences or simply ask the user directly at runtime. Less intrusive methods of interacting with the user for evaluation functions could include the user setting a "mood" switch throughout the day which may allow for the selection of mood appropriate interaction components.

3.3 Making a Choice

To choose the interaction components to use, it is necessary to take the results from the evaluation functions and decide which possibility is the best overall choice. In the example we showed a simple numerical approach to determine the best worst-case choice to use. Further numerical approaches are possible; such as determining the choice with the maximum average evaluation result, the maximum absolute value from any evaluation function or by relative rankings.

However, there are other possible approaches available. A more advanced approach could treat the choice itself as yet another evaluation which takes the results from a set of other evaluation functions. Such a model would result in a tree structure where evaluation functions can be combined in interesting ways to allow greater control over the selection of interaction components.

3.4 Graphing Possibilities

Representing dependences between components within a system is naturally accomplished by using a graph with directed edges representing the dependency.

An example of this dependency modelling was shown in our example where dependences of the concrete user interaction component were shown by a connection to the abstract user interaction that it implements. In other implementations of this approach the graph may represent data type compatibilities or other information required to determine if two components can be used together. Once the graph has been created it is possible to then traverse the graph to create a list of possibilities.

Creation of this graph can be accomplished entirely automatically through the provision of service discovery systems which are aware of which components are available in a system but other modes of use are available as well such as direct user entry of a graph.

4 Implications for Architecture Design

In the examples within this paper we have discussed the model primarily in context of the Arch model and an example of web layout algorithms. However, the evaluation model does not rely on this particular architectural model or application domain. The evaluation function based model presented here is particularly suitable for implementation within a component based system; in which case it could be represented as an "Interaction Manager" component which is capable of assigning interaction components to tasks.

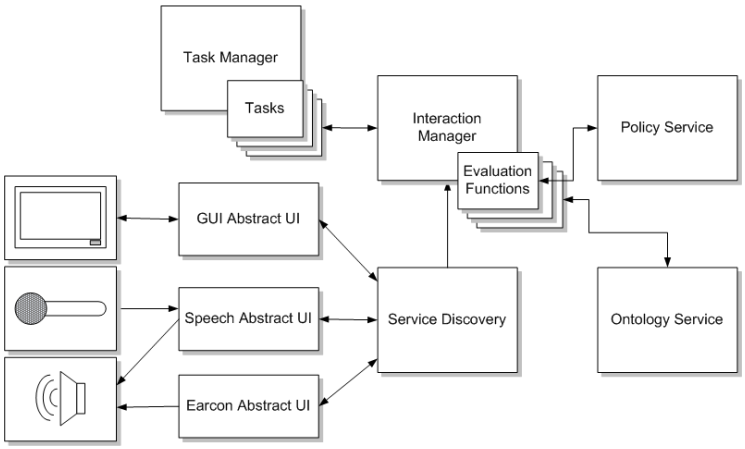

Fig. 7. Sample Architecture

Figure 7 shows a sample architecture implemented within the MATCH homecare framework[8]. Within this sample architecture there are a number of physical devices which are each represented by an abstract UI component. These interaction components are monitored by the service discovery component which

maintains a list of available devices within the system which allows for the resources (in this case devices) to change at runtime. Task represents the action to be undertaken. For example, a task might have the role of alerting a resident to a dangerous change in house state. This will result in the task object requesting from the interaction manager a mapping to one or more interaction objects by which the alert can be communicated to the resident The interaction manager then executes the model presented here using a set of evaluation functions. The evaluation functions can in turn request information from 3rd party services, in this case Policy and Ontology Description services. The ability to change evaluation functions at runtime allows for dynamic change of criteria for deciding interactions and level of human judgement.

5 Related Work

Speakeasy[9] 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. This approach is called the recombinant computing approach by its authors. Within this approach it is the users responsibility to manually perform any configuration tasks whenever they are required as well as choosing the appropriate components for the current context.

SUPPLE[10] 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. This means that each widget must have an assigned "cost" which means that SUPPLE cannot take into account any changes in costs across changing contexts.

Comets[11] (COntex of use Mouldable widgETs) is a system that provides run-time adaptation of interaction 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.[12] presented a system where the configuration of output sources was controlled by a simple "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[13] reference architecture is split into three levels of abstraction - the Interactive Systems Layer, the Distribution-Migration-Plasticity layer (DMP) and the platform layer. In the context of this work we are most interested in 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 configurator 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".

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 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.

6 Conclusion

We have shown that there is a need for interactive systems which are capable of evolving their interaction techniques subject to context and shown that there is a significant lack of approaches to do this in a generic fashion - where current state of the art techniques concentrate on a single technique. This topic has been identified as an area of interactive systems research which has yet to be fully explored.

To address this gap in the literature we have presented a model to support evolution by assisting in the choice of interaction techniques, this model is expected to be capable of bridging many different types of evaluation technique to direct selection of interactions, generally within interactive systems, but specifically within ubiquitous and homecare systems.

The primary contribution of this work is the notion that evaluation functions can be used to model requirements or preferences within an interactive system and a laying of foundations for further research into general approaches for decision making in interactive systems.

7 Acknowledgements

This research has been carried out within the MATCH (Mobilising Advanced Technologies for Care at Home) Project funded by Scottish Funding Council (grant HR04016).

References

1. Dourish, P.: Developing a Reflective Model of Collaborative Systems. ACM Transactions on Computer-Human Interaction **2**(1) (1995) 40–63
2. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable systems: pressing the issues with buttons. Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people (1990) 175–182
3. O'Brien, J., Rodden, T.: Interactive systems in domestic environments. Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques (1997) 247–259
4. Rodden, T., Benford, S.: The evolution of buildings and implications for the design of ubiquitous domestic environments. Proceedings of the conference on Human factors in computing systems (2003) 9–16
5. Bass, L.: Metamodel for the Runtime Architecture of an Interactive System. The UIMS Tool Developers Workshop. SIGCHI Bulletin **24**(1) (1992)
6. MacKay, B., Watters, C., Duffy, J.: 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 (2004)
7. Schmidt, A., Beigl, M., Gellersen, H.W.: There is more to context than location. Computers & Graphics **23**(6) (1999) 893–901
8. Gray, P., McBryan, T., Martin, C., Gil, N., Wolters, M., Mayo, N., Turner, K., Docherty, L., Wang, F., Kolberg, M.: A Scalable Home Care System Infrastructure Supporting Domiciliary Care (2007)
9. Edwards, W.K., Newman, M.W., Sedivy, J., Smith, T., Izadi, S.: Challenge: Recombinant Computing and the Speakeasy Approach. In: MOBICOM'02 - The 8th Annual International Conference on Mobile Computing. (2002) 279–286
10. Weld, D.S., Anderson, C., Domingos, P., Etzioni, O., Gajos, K., Lau, T., Wolfman, S.: Automatically personalizing user interfaces. IJCAI03, Acapulco, Mexico, August (2003)
11. Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A.: Towards a new generation of widgets for supporting software plasticity: the "comet". Preproceedings of EHCI/DSV-IS **4** (2004) 41–60
12. Crease, M., Brewster, S.A., Gray, P.: Caring, Sharing Widgets: A Toolkit of Sensitive Widgets. Proceedings of BCS Human-Computer Interaction (HCI'2000) (2000) 257–270
13. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In: Second European Symposium on Ambient Intelligence, Eindhoven (2004) 291–302