

A Framework for Runtime Evaluation, Selection and Creation of Interaction Objects

Tony McBryan and Philip Gray

Department of Computing Science

Sir Alwyn Williams Building

Lilybank Gardens, University of Glasgow, G12 8QQ, UK

+44 141 330 3541

{mcbryan,pdg}@dcs.gla.ac.uk

ABSTRACT

This poster presents an approach particularly suitable within interactive systems where selection of appropriate interaction components cannot reasonably occur until runtime and where criteria for choosing cannot be determined in advance. This approach promotes loose coupling of program components by allowing the choice of component as well as the method of choice of component to be chosen at runtime. We argue that this approach improves system architecture, maintainability, flexibility and extensibility.

Keywords

Evaluation function, interaction object, context, selection

INTRODUCTION

In this poster we describe an approach for the selection and creation of specialised interaction objects. This is particularly useful in interactive systems where objects may be entities with specialisation in man-machine interaction methods (*interaction objects*).

In interactive systems it is often more difficult to decide which interaction method to use than to actually instantiate and deploy it. For example, we can imagine a ubiquitous interactive system which can communicate with users via a variety of methods (speech synthesis, GUI, vibration of a physical device, etc.) and may have a number of different options for employing these methods (e.g. male vs. female speech synthesiser, different physical speaker locations).

Our approach addresses issues of Plasticity [1] by building upon the approach described in McBryan [2]; representing the choice of interaction explicitly within a formal framework that allows for the decision process to take place alongside the creation process of the objects. This has the further advantage of localising common functions necessary for the discovery, choice and creation processes.

APPROACH

The basic concept of the approach we introduce here involves an specialized component known as an *Evaluation Function*. The purpose of an Evaluation Function is to encapsulate the discriminatory logic involved in selecting which of the set of potentially available interaction objects are suitable in the current situation.

Evaluation functions are, essentially, arbitrary logic which are responsible for making the decisions between which, if any, interaction objects should be instantiated into actual objects. Examples of Evaluation Functions are Utility functions [5], Policy rules [6] and Context Sensitive rules [4] as well as automatic or semi automatic approaches such as the Collaborative Filtering [3].

Once a list of possible objects has been obtained it is necessary to decide which of these objects should, in fact, actually be used. This is not necessarily a single object as it may be advantageous to provide multiple interaction objects that can be used concurrently (in the case of selecting multimodal interfaces). To make this decision the Evaluation Function is queried with the list of available interaction object descriptions and expected to return a similar list containing descriptions of those interaction objects that should actually be used.

Once this list of acceptable descriptions has been obtained we employ the use of a Builder class to actually transform descriptions into their appropriate instantiations. This can be performed with a single Builder object or a collection of Builder objects which are delegated to by the main Builder object.

Combining these components we can deliver an approach which is capable of:

- Performing Component Discovery
- Discriminating between Interaction Object Descriptions using Evaluation Functions
- Allowing the choice of criteria for discrimination to be altered by selecting the Evaluation Function to be used
- Presenting a more consistent and formalized approach to the selection of interaction objects

- Representing criteria for discrimination as live objects; allowing these criteria to be manipulated and contain state relating to the decisions

The result of this high level logic is to provide a collection of interaction objects in the products collection which can be used by the directing application as it requires.

HANDLING COMPLEX COMPONENT EVALUATIONS

A need to compose and manipulate evaluation criteria to create more powerful and more useful functions is naturally supported in this approach. Since an Evaluation Function is essentially a method call and has a well defined interface it is possible for one Evaluation Function to call other Evaluation Functions as helper or sub-functions.

This allows for and encourages modularity in Evaluation Functions which makes them simpler and easier to compose. By making modular functions we also open up the route to allowing user composition (as well as advanced user creation) of Evaluation Functions which can increase the users perceived flexibility of an interactive system.

To combine Evaluation Functions we can take the approach of treating Evaluation Functions as a tree structure. The root of this tree is responsible for accepting a set of available options and is responsible for delegating and/or selecting between its child Evaluation Functions in the tree.

Figure 1 demonstrates a tree structure which is composed of five Evaluation Functions. At the root of this tree is a Context Sensitive function which selects based on some element of context between its children; the Union and Utility functions. In this case the Context Sensitive function may be dependant on the importance of the message. For low priority messages the left branch is chosen (allowing the users preferences to decide) and otherwise the right branch is chosen (a utility function that minimises the latency until message receipt). The Union function queries both its children (each representing a single user's preferences in a multi-user environment) and performs a logical union on the results.

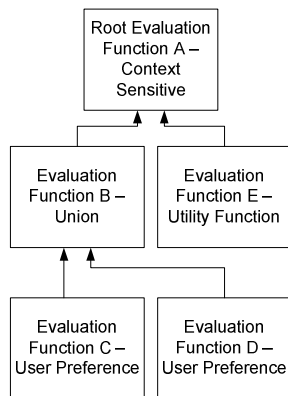


Figure 1: An Example Composition of Functions

We have developed a simple language for describing this tree structure which can be concisely described in Code

Listing 1. This language can be used by developers or users to specify how to combine Evaluation Functions.

EVALUATION_FUNCTION_NAME : String Literal
identifying the program code that represents the function

PARAMETERS : Key/value mapping of parameters for an Evaluation Function

EVALUATION_FUNCTION :
<EVALUATION_FUNCTION_NAME> (<PARAMETERS>)
{ <EVALUATION_FUNCTION>* }

Code Listing 1: Evaluation Function Combination

By supporting modular Evaluation Functions it is possible to reuse existing Evaluation Function in new ways by combining them together differently rather than having to develop new ones. This has the result of reducing the effort required to develop a range of mechanisms for selecting interaction objects.

CONCLUSION

In this poster we present an approach to the selection of interaction objects which we believe supports looser coupling of components. The approach was designed to enhance maintainability as well as allowing new evaluation methods to be added without requiring large scale redesign.

These advantages are offered by explicit modelling of Evaluation Functions which encapsulate selection logic.

ACKNOWLEDGMENTS

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

REFERENCES

1. Calvary, G., J. Coutaz, et al.. "Towards a new generation of widgets for supporting software plasticity: the "comet"." *Proceedings of EHCI/DSV-IS 4*: 41--60. (2004)
2. McBryan, T. and P. Gray. A Model-Based Approach to Supporting Configuration in Ubiquitous Systems. *Design, Specification and Verification of Interactive Systems 2008*, Kingston Ontario Canada (2008)
3. Goldberg, D., D. Nichols, et al.. "Using collaborative filtering to weave an information tapestry." *Communications of the ACM*, 35(12): 61-70, (1992).
4. Schilit, B. N., N. Adams, et al.. Context-aware Computing Applications, Xerox Corp., Palo Alto Research Center. (1994)
5. Sousa, J.P. and Garlan, D.: Improving User-Awareness by Factoring it Out of Applications. *Proc System Support for Ubiquitous Computing Workshop (UbiSys)*, (2003)
6. Wang, F., L. S. Docherty, et al.. Service and Policies for Care at Home. *International Conference on Pervasive Computing Technologies for Healthcare*, Innsbruck, Austria.(2006)