Modeling Preference for Adaptive User-Interfaces

Jacob Eisenstein
RedWhale Software
277 Town & Country Village
Palo Alto, CA 94303
jacob@redwhale.com

Abstract
The incorporation of plastic and adaptive user-interfaces into the model-based paradigm requires a new, more flexible modeling formalism. Rather than modeling the user-interface as a set of static structures and mappings, the UI should be modeled as a set of design *preferences*. Preferences are frequently many-to-one or many-to-many relationships that elude conventional UI modeling, which has largely focused on one-to-one mappings. In this paper, a spectrum of preference relations is described, and a new syntax for modeling preference is proposed. This spectrum extends from simple one-to-one *bindings* to complex *design guidelines* that can be structured together to implement decision trees. This new representation allows decision trees to be tightly integrated into the user-interface model itself, enhancing their flexibility and power.

1. Introduction

Model-based user-interface development has been characterized as a process of creating *mappings* between elements in various model components [3]. For example, interactor selection can be thought of as finding a set of appropriate mappings between abstract elements in the domain model and widgets from a presentation model. Typically, these mappings are assumed to be "one-to-one", connecting two single elements together. However, many important user-interface design relationships require many-to-one or even many-to-many mappings.

While a general-purpose framework for representing such mappings might be desirable for the flexibility that it offers, such a framework would add considerable complexity to the representation language and impose a significant burden on user-interface modeling tools. One specific phenomenon that commonly eludes description by one-to-one mappings is *preference*. Preference relations allow for a more flexible and adaptable specification of the user-interface; rather than specifying exactly how the UI must appear, the designer can specify what *would be* preferred, and in which situation. This freedom is particularly important for user-interfaces that must run on heterogeneous devices [4], since the contexts of use vary and may even change at run-time. Preference modeling is also critical if interaction is to be dynamically customized for the user; preference relations can be used to show how the user-interface should change in relation to the user model. In this paper, a formalism for modeling preference is proposed.

2. A Spectrum of Preference Relations

Preference relations range from simple one-to-one bindings to sophisticated design guidelines. This spectrum of preferences can be divided into two broad classes: concrete preferences and abstract preferences. Concrete preferences specify their targets directly, although an ordered list of targets is permitted. Abstract preferences specify their targets indirectly, based on *criteria* that describe characteristics of either the targets themselves or some other *object*.

	Condition	Target	Criteria	Object
Binding	1	1	None	None
Simple Preference	Any number	1	None	None
Ordered Preference	Any number	Any number	None	None
Abstract Preference	Any number	More than 1	Criteria apply to targets; logical and preferential criteria are allowed	None
Design guideline	Any number	Any number	Criteria apply to object; only logical criteria are allowed	Any number

2.1 Concrete Preferences

In any preference relation, there is first a set of *conditions*, under which the preference becomes valid. For example, if user U1 prefers for presentation element P1 (say, a listbox) to represent domain object D1, then U1 and D1 are the conditions. P1 is the *target* of the preference relation—the thing that is preferred. Every preference relation must have at least one condition and one target. A preference relation that involves only one condition and one target will be referred to as a *binding*. A preference relation that involves any number of conditions and a single target will be referred to as a *simple preference*. A designer might also want to enumerate a set of interactors, in order of preference. In this case, there are multiple targets. Preferences that involve multiple conditions and multiple targets, are referred to as *ordered* preferences.

2.2 Abstract Preferences

Bindings, simple preferences, and ordered preferences are all *concrete* preferences, because the targets are specified directly. Rather than specifying the element that is preferred, a designer may wish to instead specify the *characteristics* of the element that is preferred. For example, a designer may prefer whatever presentation element requires the least number of clicks, or whatever dialog structure imposes the least cognitive load. This kind of preference is referred to as an *abstract* preference.

Abstract preferences have an additional feature—a set of *criteria*. The criteria determine which target is selected; in the example above, the number of clicks required is the criterion. For an abstract preference, the targets are possible selections, and criteria determine which target is actually chosen. Needless to say, an abstract preference is required to have more than one target. There are two types of criteria: preferential and logical. Preferential criteria specify those characteristics that cause one target to be *preferred*. For example, a preferential criterion might say to choose the dialog structure with the least complexity. Conversely, logical criteria specify what kinds of targets are *allowed*. A logical criterion might say not to choose any dialog structure with complexity greater than 50. If there is more than one preferential criteria, then each criterion must have a *priority* to indicate how important it is relative to the others. Logical criteria need no priority; if the value of the feature under consideration by a logical criterion is in violation of the criterion, then the associated target is ruled out.

When used in combination, preferential and logical criteria can allow for very complex preference specifications. For example, consider the following specification: "For task model Tm2 and domain model Dm1, user U4 likes presentation elements that require few clicks, take up little screen real estate, and have lots of colors. The criteria of having lots of colors is most important, followed by the number of clicks. But if amount of space occupied is too small, it won't be visible, so disallow it altogether." In this situation, Tm2, Dm1, and U4 are the conditions. The set of presentation elements under consideration are the targets. There are three preferential criteria: number of colors (more is preferred), number of clicks (less is preferred), amount of space occupied (less is preferred). In addition, there is one logical criteria: the amount of space occupied must be greater than some constant. For a table describing this preference relation, see the second scenario in section 3.1.4.

2.2.1 Design Guidelines

Thus far, it has been assumed that the criteria apply only to the targets. That is, distinctions can be made between the targets only on the basis of characteristics of the targets themselves. It is possible to say, for example, "U4 prefers the least complex task tree," but it is not possible to say, "If U4's experience level is less than *intermediate*, then use presentation element P3." To do this, we need to separate the list of targets from the criteria.

Preference relations where the criteria do not necessarily apply to the targets are called *design guidelines*. Design guidelines are the most abstract and complex preference relations that we will attempt to model. For this class of preference relations, an additional feature must be considered: a list of objects. The criteria refer to features of each object. For design guidelines, all criteria must be logical. If the value of every criteria is true for every object, then a mapping is created between the conditions and each of the targets.

Recall the example above: "If U4's experience level is less than intermediate, then use P3." In this case, U4 is the object. The criterion is the "experience level" attribute. The target is P3. If the experience level meets the criteria of being less than intermediate, then the target P3 is chosen and the mapping is made.

The target of a design rule could itself be another preference relation. For example, suppose we wanted to model the following preference: "if U4's experience level is less than intermediate, then use the task tree with the least amount of complexity." Once again, U4 is the object, and experience level is the criterion. But the target is a preference relation – specifically, an *abstract preference.* That abstract preference says to choose the task tree with the least amount of complexity. So for the abstract preference, the target is a list of task trees and the criterion is the amount of complexity. The condition of the abstract preference is simply the design guideline that targeted it.

By creating a series of *design guidelines* that target each other, we can implement a decision tree to represent complex abstract preferences. Decision trees have already been used for representing UI design guidelines [5]. However, never before has it been possible to integrate a decision tree so tightly into the user interface model. This offers a number of important advantages. Whereas the set of discriminants is static in all known previous applications of decision trees to UI design, this new formalism permits any attribute defined in the user interface model to act as a discriminant. Moreover, rather than limiting use of the decision tree to one particular kind of mapping—e.g., interactor selection, as in [5]—any type of mapping can be performed. Decision trees can just as easily be used to select among alternative dialog structures or even color schemes.

To see how this can work, let us consider an example in which a decision tree selects interactors by first considering the type of a domain element D1, and then the experience level of the user U1. We discriminate based on the type of D1 (e.g. float, integer, string) by creating a set of design guidelines $G_1 \ldots G_N$: one for each possible type. Each such guideline has U1 and D1 as conditions. D1 serves as the object, and the type of D1 is the criterion. For G_1, the criterion is met iff the type of D1 is "integer." For G_2, the criterion is met iff the type of D1 is "string," and so on. If the criterion is met, then D1 and U1 are mapped on to the targets; otherwise, no mapping is made. In this case, the targets are another set of design guidelines. For G_1, those targets are design guidelines $G_{1.1} \ldots G_{1.M}$. For G_2, those targets are design guidelines $G_{2.1} \ldots G_{2.M}$. Thus, if the type of D1 is "integer," then D1 and U1 are mapped on to the design guidelines $G_{1.1} \ldots G_{1.M}$. These design guidelines must then be evaluated.

If the criterion is met for at least one of $G_1 \ldots G_N$, then the evaluation of the decision tree continues. $G_{1.1}$ is structured similarly to G_1. The condition of $G_{1.1}$ is now G_1, and the object is now U1; the target is an interactor. The criterion is the experience level of U1. For $G_{1.1}$, the criterion is met iff the experience level of U1 is "advanced." For $G_{1.2}$, the criteria is met iff the experience level of U1 is "intermediate," and so on. Each of $G_{1.1} \ldots G_{1.n}$ represents a user level, and each has a target P, which is an interactor. If the criterion is met, then G_1 is mapped on to P. Mappings of this kind are transitive. If U1 and D1 are mapped onto G_1, and if G_1 is mapped onto the interactor P, then U1 and D1 are mapped onto P. The design guidelines are evaluated in a breadth-first manner; once all guidelines are evaluated, the mappings (if any) are returned. We are then finished evaluating the decision tree.

The tight integration of the decision tree with user-interface model offers significant advantages over previous design guideline implementation strategies. However, it poses a problem of model binding. In the example decision tree, it is assumed that the relevant design guidelines have been created with the appropriate objects and conditions, which are themselves parts of the user-interface model. In reality, this is not likely to be the case, because decision trees of design guidelines are usually created independent of any specific user-interface model, and are intended for reuse among several user-interface models. The solution is to underspecify the decision tree, leaving the objects and conditions empty. When the decision tree is applied to a specific interface model, a binding procedure for linking up the appropriate objects and conditions is necessary. Of course, some kind of binding must be performed whenever a decision tree or any other set of general design rules is applied to a specific user-interface model.

3. Implementing Preference Relations in XIML

XIML, the eXtensible Interaction Markup Language, is an XML-based user-interface modeling language, described in [1,2]. This formalism for modeling preference relations has been incorporated into XIML as part of the Design Model Component. The design model consists of a list of preference elements. A preference element can have four child elements: *conditions*, *targets*, *objects* and *criteria*. Each of these elements contains a list of *relation statements*, which indicate a mapping to another element somewhere else in the UI specification. Relation statements are simple one-to-one mappings, with a reference to an element's ID and a semantic definition. Relation statements can also include little bits of information, which are specified in *attribute statements*.

All criteria demand three common attribute statements: the name of the criteria (e.g. "screen space", "user experience level"), the type of the criteria ("preferential" or "logical"), and the behavior. The behavior of preferential criteria can take the following values: *minimize, maximize, approach, retreat*. If either of the latter two values is taken, then a *threshold* must be supplied; this is the value that the designer is trying to approach or retreat from. Preferential criteria also have a *priority*, which specifies the importance of the criterion relative to other criteria. The example in section 3.1.4 should make this clearer. The behavior of logical criteria can take the following values: *greater than, less than, equals, not equals*. All logical criteria must have a *threshold*. Logical criteria do not have a priority; if all criteria are met then the mapping holds; otherwise it does not.

3.1 Examples

This section offers examples of how to model various preference phenomena. All examples have also been modeled in XIML code; please contact the author for more information.

3.1.1 Binding

Scenario: "User U1 prefers presentation element P1."

CONDITIONS	TARGETS	OBJECTS	CRITERIA
U1	P1		

3.1.2 Simple Preference

Scenario: "User U1 prefers presentation element P1 for representing domain model Dm1."

CONDITIONS	TARGETS	OBJECTS	CRITERIA
U1, Dm1	P1		

3.1.3 Ordered Preference

Scenario: "User U1 prefers presentation element P1 to presentation element P2, for representing domain model Dm1."

CONDITIONS	TARGETS	OBJECTS	CRITERIA
U1, Dm1	P1 – priority 100 P2 – priority 50		

3.1.4 Abstract Preferences

Scenario: "User U1 prefers the presentation element that occupies the least screen space, but not less than 100 square pixels."

CONDITIONS	TARGETS	CRITERIA		OBJECT
U1	P1, P2,…Pn	**Name**	Screen Space	
		Type	Preferential	
		Behavior	Minimize	
		Priority	100	
		Name	Screen Space	
		Type	Logical	
		Behavior	Greater Than	
		Threshold	100	

Scenario: "For task model Tm2 and domain model Dm1, user U4 likes presentation elements that require few clicks, take up little screen real estate, and have lots of colors. The criteria of having lots of colors is most important, followed by the number of clicks. But if the amount of space occupied is too small, it won't be visible, so disallow it altogether."

CONDITIONS	TARGETS	CRITERIA		OBJECT
Tm2, Dm1, U4	P1, P2,…Pn	**Name**	Screen Space	
		Type	Preferential	
		Behavior	Minimize	
		Priority	25	
		Name	Screen Space	
		Type	Logical	
		Behavior	Greater Than	
		Threshold	100	
		Name	Clicks	
		Type	Preferential	
		Behavior	Minimize	
		Priority	50	
		Name	Number of Colors	
		Type	Preferential	
		Behavior	Maximize	
		Priority	100	

3.1.5 Design Rules

Scenario: "If user U1 is *not* of experience level *expert*, then select the presentation element with the least cognitive complexity."

Design Rule G1

CONDITIONS	TARGETS	CRITERIA		OBJECT
U1	G2	**Name**	Experience Level	**U1**
		Type	Logical	
		Behavior	Not Equals	
		Threshold	Expert	

Abstract Preference G2

CONDITIONS	TARGETS	CRITERIA		OBJECT
G1	P1, P2,…Pn	**Name**	Cognitive Complexity	
		Type	Preferential	
		Behavior	Minimize	
		Priority	100	

References

1. J. Eisenstein. “XIML: The eXtensible Interaction Markup Language.” RedWhale Software internal document. Please contact author for additional information.
2. A. Puerta, and J. Eisenstein. “A Representational Basis for User-Interface Transformations.” 2001 CHI Workshop on Transforming the UI for Anyone, Anywhere. Seattle WA, March 2001.
3. A. Puerta and J. Eisenstein. “Towards a General Computational Framework for Model-Based Interface Development Systems.” Knowledge-Based Systems, Vol. 12, 1999, pp. 433-442.
4. D. Thevenin and J. Coutaz. “Plasticity of User Interfaces: Framework and Research Agenda”, in Proceedings of INTERACT’99. Edinburgh: IOS Press 1999.
5. J. Vanderdonckt and P. Berquin. “Towards a Very Large Model-Based Approach for User Interface Development”, Proceedings of UIDIS’99. Los Alamitos: IEEE Press 1999, pp. 76-85.