Abstract

The purpose of user interfaces in modern applications is to assist users in specifying parameters to execute a set of commands. It is common to find interdependencies among these parameters. As a result, user interfaces in general are often complex, incomplete, or difficult to implement. Property models attempt to alleviate this by increasing reuse in user interface programming. Property models employ a constraint system and reusable generic algorithms to do away with ad-hoc “event handlers”. Until now, property models have been applied only within desktop applications. This paper describes how the property model approach can be extended to web applications.

The primary goal of my research is to determine the best architecture for running property model based user interfaces over the web. To facilitate my research, I created a Rapid Application Development (RAD) framework to allow convenient development and testing of such interfaces. This framework constructs a user interface from the following: (1) a declarative specification of a property model, and (2) a layout description. Any change in the property model or the layout description is immediately reflected in an interactive user interface generated by the framework.

As a result of this project, it is now possible to conduct experiments on property model based user interfaces in web applications. I hope to show that adapting property models to the development of web applications will result in increased reuse and reduced defects found within an application. Furthermore, interfaces with similar functionality cannot easily be created from existing implementations.

As in other areas of computer science, we can employ reuse as an alternative to such ad hoc solutions. A standard library is an example of such reuse: libraries of data structures and algorithms are written (correctly) once and used many times. Similarly, toolkits with reusable widgets are common in modern web interfaces. The aim of these toolkits is to aid programmers in developing user interfaces by applying reuse to the individual components. While this is helpful to the programmer, the composition of components remains ad hoc.

Property models [1] have been developed as a means to model these sometimes complex behaviors. Property models employ a constraint system as a reusable model for the composition of user interface components. Through the use of a constraint system, property models can be very complex, as demonstrated by Figure 2. These complex networks commonly appear in user interfaces and can be difficult to express by traditional event handlers. Property models capture these relationships by using a constraint system. A constraint is simply a relationship between variables that needs to be upheld. As demonstrated in Figure 3, constraint systems consist of two parts: (1) a set of variables tied together by constraints, and (2) methods that enforce these con-

1. Introduction

The purpose of user interfaces in modern applications is to assist users in specifying parameters to execute a set of commands. An interface might support various behaviors, including computing the values of some interface elements automatically when values of other elements change, storing and retrieving default values, capturing user actions into a replayable script, undo and redo functionality, disabling user interface elements when their values are irrelevant for a final result, etc. Correctly supporting these behaviors is difficult and implementation attempts often lead to many of the defects found within an application. Furthermore, interfaces with similar functionality cannot easily be created from existing implementations.

Categories and Subject Descriptors D.2.2 [Design Tools and Techniques]: User interfaces; D.2.13 [Reusable Software]: Reuse models

General Terms Algorithms

Keywords Software reuse, Component software, User interfaces, Declarative specifications, Constraint systems

As mentioned earlier, a property model is a tool for increasing reuse in user interfaces. It does this by capturing complex relationships among components and making them explicitly available to algorithms. A variety of user interface behaviors can be expressed in a general and reusable manner as algorithms over a property model.

Consider a dialog for resizing an image that might appear in an image-editing application. An example of such a dialog appears in Figure 1. The network of relationships among the components can be very complex, as demonstrated by Figure 2. These complex networks commonly appear in user interfaces and can be difficult to express by traditional event handlers. Additional components can significantly affect the complexity of the system.

A property model captures these relationships by using a constraint system. A constraint is simply a relationship between variables that needs to be upheld. As demonstrated in Figure 3, constraint systems consist of two parts: (1) a set of variables tied together by constraints, and (2) methods that enforce these con-
constraints. The act of executing a method satisfies a constraint, by computing new values for some subset of variables in the constraint from the values of the other variables in the constraint. A solution to the constraint system can be expressed as a method evaluation order ensuring that constraints are correctly satisfied, such that no variable is assigned by two different methods. In overly constrained systems, weaker constraints can be discarded until a solution is formed.

A property model alone is not enough to build a user interface, however. A layout description, that includes bindings between user interface elements and variables in the model, is necessary to determine the appearance of the interface. This clear separation between the data model and presentation allows the programmer to easily specify several interfaces driven by the same property model. Additionally, interface widgets can be stateless and thus easier to define.

3. Approach

In order to determine the best architecture for running property model based user interfaces over the web, I created a system for conveniently developing and testing such interfaces. Prior work had yielded a property model solver. This solver accepts two inputs: (1) a declarative specification of the property model, and (2) a priority order. A RAD framework allowed iterative development of a system that could incorporate the preexisting solver.

4. Experience

The implementation of a RAD environment requires a few details that enable it to perform efficiently and securely. For security, a web browser will not allow any command line code to run on a client computer. In this case, in order to use the solver with my application it needed to be placed on the server.

A client/server relationship commonly experiences a slow down when transferring data from the client to the server or vice versa. To speed up an application with this relationship, it is optimal to have most of the processing within the client’s browser. As scripts are entered into the RAD environment hidden processes take place to give the illusion of instant processing. When the layout script is updated, processes within the client’s browser are executed to update the interactive live view of the generated user interface.

For speed and security reasons, the same is done with foreign function calls. As these functions are entered into the script of the web application they are dynamically placed into the web page itself. This allows the functions to be executed from within the generated user interface.

When the declarative specification of the property model is edited, however, it must be sent to the solver on the server. This can potentially slow down the application if a large amount of data is sent. A web based property model solver can alleviate this problem.

5. Conclusions

The current state of the application allows for the convenient development and testing of property model based web interfaces. Due to the recent completion of the application, there are no results to report on property model based web interfaces.

6. Future Work

Previous research has shown the property model approach to be an improvement over traditional "event handling" methods in desktop applications. [1] Testing to prove that this is also true for property model based web interfaces has yet to be completed.
documentation and a number of examples demonstrating property model based web interfaces are also in development.

References