Picture Programming Project

Nita Goyal, Charles Hoch, Ravi Krishnamurthy
Brian Meckler, Michael Suckow, Moshe Zloof

Hewlett-Packard Laboratories, Palo Alto, CA
lastname@hpl.hp.com

Introduction

If Picture is worth a thousand words, why are we still writing programs using words? This is the lofty guiding principle for the ICBE system being developed by Picture programming project at HP Labs. The goal of ICBE system is to build most (database) applications pictorially and declaratively. Furthermore, end users with no programming background are expected to do this pictorial programming. We have currently implemented a subset of this goal that pertains to the problem of presenting data for subsequent browsing and interaction. Such presentation for browsing and interactions is termed *rendering* of data. The rendering applications have gained new importance in the context of World Wide Web, where not only large amounts of data need to be rendered but also the programming knowledge of a typical web publisher is quite rudimentary. Therefore,

> *the goal of the current system is to allow a web publisher with rudimentary programming background to build rendering applications that can be embedded as an applet in a web page.*

In today's technology, each application renders the data from a database in an application-dependent manner, typically programmed using some GUI builder and traditional programming language. The semantics of human-computer interactions with the data (albeit, application specific), is hard wired into the program and typically redone for every new application. To address this problem, we proposed a declarative language called Rendering By Example (RBE) [KZ95], in which rendering characteristics could be stated in an application independent manner. For a large class of rendered data, a program in RBE can declaratively specify the inherently procedural interaction semantics.

A declarative program addresses the problem of application independent semantics and the reusability in many applications, but web publishers may still not be able to program in it. To assure that they can build these rendering applications, the process of constructing an application must mimic the process of using the application. We assume that anybody can use a rendering application and understand the interactions in the context of that particular application. Therefore, web publishers construct a new rendering application by starting from a sample rendering application that is actually working and is running against an actual data and then modifying it. Once the modified sample is what the web publisher wants in his/her rendering application, then the system infers the "rendering program" from that example.

This approach to programming by example is quite different from QBE wherein a representative example is stated in the context of the template (i.e., schema) of a table and that example constitutes the program; i.e., conceptually, the program in QBE is a domain calculus statement over table templates.

In this document, we explain the application independent abstractions in RBE and the WYSIWIG process of programming starting from an example.

Rendering By Example

Rendering By Example (RBE) was proposed in [KZ95] as a declarative language to express a rendering of data, where a rendering is defined as a presentation of data with subsequent browsing and interaction semantics.

A rendering of data consists of a set of screen widgets populated with data. Such rendering applications allow the browsing and interaction with the data in ways specific to that application. These application-specific rendering applications facilitate the user to find and assimilate the data. Rendering of data is widely used in most applications that have a GUI interface and therefore is not a new concept. But the novelty here is the process of constructing such rendering applications.

Traditional programming using a state of the art GUI builder would require the following steps.

1) Place the widgets on the screen; i.e., paint the screen.
2) Program each event for the widgets, wherein the semantics of the interactions are specified. These include changing data in other widgets and restricting/facilitating the choices of interactions in widgets etc.

This type of event programming deals with not only the inter-widget relationships but also has to deal with the idiosyncrasies of the individual widgets and the windowing environment. Thus, the resulting code is fairly complex. Further, the semantics of interacting with a widget can potentially affect the data in many other widgets. As a result, the programming complexity grows non-linearly with the number of widgets on the screen.

In contrast, a rendering application written in RBE (i.e., rendering program) is a declarative statement of the rendering. We propose abstractions that can be used to specify any rendering program so that the behavior due to interactions in the rendering can be expressed. A rendering program is stated in a pictorial fashion by placing the widgets on the screen and binding the widgets to the attributes in the database.

The effects of an interaction are observed within the widgets involved in the interaction as well as in other widgets not involved in the interaction. Without loss of generality[1] let us consider the effects of user interaction with a single widget. So the effects of an interaction can be categorized as intra-widget and inter-widget depending on where the changes occur.

- Intra-widget: These are effects of interactions with a widget that are within the same widget. Dropping down a pick-list, choosing one element in a radio button group, multi-selecting elements of a list/table are examples of intra-widget effects.
- Inter-widget: These are effects that change widgets other than the interaction widget. Examples are changing the data shown in other widgets and restricting the access to data in other widgets.

Note that an interaction can have both intra- and inter-effects and each of these effects may transitively have other effects.

Intra-widget effects can be abstracted by categorizing the widgets by a set of properties. Examples of properties are set/singleton representability, duplicates representability, value selectability, multi-value selectability etc. Using these properties of widgets not only can the intra-widget behavior be characterized but also the inter-widget behavior can be implied. For example, if a set of values is rendered in a text-box, which usually can only represent a singleton value, then the system must provide some means to navigate to other values in the data such as back/next buttons. Further, extending the system to include new widgets can be made semi-automatic using these widget properties. As a result of this widget categorization, a rendering program has been defined to have appropriate meaning for all possible combinations of properties. In other words, the semantics of a rendering is not based on the individual idiosyncrasies of any widget such as text-box or radio button but of some generic properties. We conclude from this observation that any widget in a given rendering can be replaced by any other widget and the "meaning" of the new rendering can be appropriately defined; i.e., the system will manage the intra-widget and inter-widget behavior appropriately.

Inter-widget effects in any rendering are typically localized by some grouping of widgets that are either visually or semantically obvious to the user. Sub-forms and two-level forms are examples of this type of grouping. Such groups are necessary because if every widget can potentially affect every other widget on the screen then the rendering is likely to become incomprehensible. But group of widgets can affect other groups, which means that the same incomprehensibility problem for widget can also affect the groups, if there are lots of groups. In order to avoid this recurrent problem, groups are typically hierarchically ordered in most renderings and groups can only affect the groups below and not vice versa. Such a hierarchical decomposition of the set of widgets on the screen is an assumption widely used in rendering applications. We make the same assumption and term each group in the hierarchy a *level*. The rendering semantics can be recursively defined based on the hierarchy and the semantics of a single level. Note that specifying the levels can be done graphically or by declaring the parent of each widget.

Inter-widget effects within a single level can be categorized into three categories.

1) Synchronized Level
2) Dynamic Level
3) Anchored Level

These three categories of levels dictate the behavior of the widgets with respect to each other in the level. The specific meaning of each is beyond the scope of this write-up. It can be argued that these three categories are comprehensive for most renderings. Thus, the inter-widget effects of any interaction can be expressed by declaring the category of the level. Note that a category of any level can be changed to another and the net effect is to give a new behavior amongst the widgets in the level. This is true irrespective of the set of widgets in that level.

In summary, a rendering program is constructed as

[1] Effects due to interaction with more than one widget can be generalized as the union of individual effects.

follows:

1) Place the widgets on the screen; i.e., paint the screen.
2) Hierarchically partition the set of widgets on the screen.
3) Declare the category of each level.
4) Bind the widgets to the attributes from the database.

This results in a rendering program that is devoid of events programming and other such procedural coding.

Even though these steps may seem straightforward and simple, it is still a daunting task for an user to conceptualize their rendering applications in terms of the above abstractions such as level categories, widgets classifications etc. We address this problem next.

WYSIWIG Programming Environment

There are two major problems with programming:

1) Where to begin? This is the age-old writers' block problem faced by any author. The web publisher has the same problem in deciding where to begin in constructing the rendering program.
2) Indirect programming: Programming has always been indirect in the sense that the execution of the program is the ultimate goal and a program is an indirect statement of that ultimate goal. Consider text with HTML tags that represents a web page. Obviously editing/constructing this HTML file is an indirect specification of the ultimate web page. In contrast, a WYSIWIG web page editor is a direct manipulation of that page and a vast improvement over editing the HTML file.

In our current implementation, we have addressed these two problems to enable web publishers to construct rendering applications easily.

We address the writers' block problem by allowing the web publisher to peruse through large selection of samples and choose one of them to modify. Each sample is a running rendering application, using actual data, which enables the web publisher to test run that rendering. Further, it is likely to be easier to modify an existing application than constructing one from scratch, if

a) the starting application is similar to the one web publisher has in mind; and
b) the modifications are easy to do.

The former is assured by providing a large sample base that can be extended by the web publisher's own rendering applications and the latter is achieved by performing the modification to the running application in a WYSIWIG fashion.

WYSIWIG modification of a running application is the process of changing the rendering application to another rendering application. This is very similar to the spreadsheet programming wherein the formulae are edited directly into a spreadsheet that is a running application and the effect of the modification are immediately seen. In a rendering application, the repertoire of WYSIWIG modification includes changes to the interactions with the widgets. If each such modification results in a new rendering application that also works and can be "test run", then the web publisher can make incremental changes until the final program matches exactly what they have in their mind. The well-known advantages of WYSIWIG editing are the direct specification of the changes to the application, immediate feedback of those changes and the ability for trial and error. All of these are very useful advantages that enable the web publisher to construct the rendering application that behaves exactly in the manner one wishes. Further, it also facilitates making incremental modification to the rendering application to cope with future needs.

The WYSIWIG modus operandi is possible because every rendering program can be mutated by a sequence of modifications to any other rendering program. This reachability property is mainly due to the fact that the declarative specification of level categories and widget categories allow all combinations with appropriate semantics for interactions. Therefore, mutating by changing level/widget categories as well as adding new levels/widgets ensure that any rendering application can be constructed through a series of mutations.

ICBE System

Rendering is but a part of the entire ICBE language. ICBE has many other abstractions so that general applications can be expressed. Applications in ICBE is viewed as the customization of the user-interface and the system, where the system customization is done by interoperating with existing databases, applications and other business objects.

In order for ICBE to be general, we designed the language to be extensible so that any ICBE program can be extended by escaping to a programming language that does not violate the declarative aspects of the ICBE language. To do this, we chose a domain calculus, Horn clause logic based language termed Logic++ [GHKMS 95] that integrates naturally into ICBE. As Logic++ is a complete programming language, we chose to implement the ICBE system in this language. We reported on this experience in [GHKMS 95]. In this sense, the current ICBE system is an ICBE application bootstrapped in ICBE system.

[GHKMS 95] Goyal, N., Hoch, C., Krishnamurthy, R, Meckler, B., and Suckow, M. "Is GUI Programming a Database Research Problem?", SIGMOD 96, Montreal, Canada.

[KZ95] Krishnamurthy, R., and Zloof, M. M., "RBE: Rendering-By-Example", Intl. Conference on Data Engineering, Taipei, 1995.