User-Centric Personalized Extensibility for Data-Driven Web Applications*

[Demonstration Paper]

Nitin Gupta[1], Fan Yang[1], Alan Demers[1], Johannes Gehrke[1],
Jayavel Shanmugasundaram[2]
[1] Cornell University, Ithaca, NY
[2] Yahoo!, Santa Clara, CA
{niting, yangf, ademers, johannes}@cs.cornell.edu, jaishan@yahoo-inc.com

ABSTRACT

We describe a novel programming model for building, extending, and personalizing web-based data-driven applications.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environment

General Terms

Design, Languages

Keywords

Database Application, Declarative Language, Extensibility

1. INTRODUCTION

An important class of applications is *data-driven web applications*, i.e., web applications that use a back-end database system. Examples of such applications include numerous B2C sites such as online shopping sites and online auctions, as well as various B2B portals. The application logic tier of a web application is normally visible to the developers of the website, while clients (or users) interact through web browsers.

Most current web applications are extremely restrictive in terms of extensibility, especially from the user's point of view. To give users the ability to make even trivial changes, for example in the visual appearance of the application or in the way data is displayed, the application developers must design and program application features to implement these changes and allow users to select among them. Thus, users can extend only what the developers predicted they might want to extend. Providing extensibility in this way requires clairvoyance on the part of the developers, and significantly complicates the structure and development process of the application.

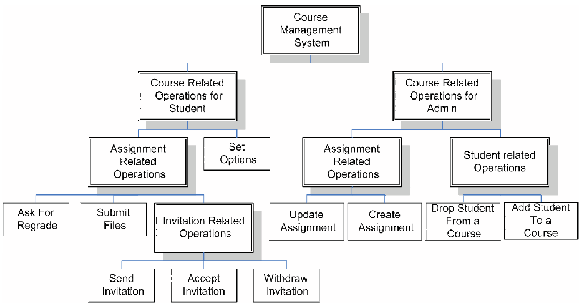

Figure 1: AUnits that model the hierarchical structure of Course Management System

Consider the following case of user-desired extensibility, which we encountered first hand. We have developed a University Course Management System for maintaining information about courses, the students enrolled in them, their assignments, etc. [3]. For each course, there is a page that shows the professor summary information about each assignment — the minimum, maximum, and average grade. Suppose, in addition to these three values, Professor X wants to see a histogram of student grades for each assignment. The only choice she has is to contact the developers of CMS, who promise to include the histogram in the next release of the system (which appears in six months), at which time the new feature becomes available to all professors.

In this demonstration, we present an extensible programming model based on a unified declarative high-level language called Hilda [3], and show how a user can add new features to a web application without having to write any code or contact the developers. We also show how a user can easily extend features from existing modules of an application to other modules, and how a developer or a user, using the Hilda GUI, can handle access permissions for the modules and data. Lastly, we demonstrate how a developer or a user can add personalized extensions to her domain in the application. The Hilda system is available as open-source software at www.cs.cornell.edu/database/hilda.

We will demonstrate the benefits extensibility in Hilda using a Course Management System (CMS) [2] used by more

*This material is based upon work supported by the National Science Foundation under Grant 0534404. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

SIGMOD'07, June 12–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

than 2000 students, staff and faculty at Cornell University. The original version of the application was developed using traditional application development tools (relational databases, J2EE, JavaScript, HTML), while a new version has been developed using Hilda.

The rest of this demonstration is organized as follows. In Section 2, we describe the Hilda language and extensibility in Hilda, and in Section 3, we describe the demo in detail.

2. HILDA AND EXTENSIBILITY IN HILDA

Hilda is a unified high-level language designed for developing data-driven web applications [3]. It is based on UML [1] and the relational data model. The language models logic for all layers in the application stack (database, application server and client side) in a unified, declarative manner.

Hilda models application logic using building blocks called *AUnits* (for Application Units), analogous to UML classes. AUnits encapsulate functionality, data and state as relational tables for each functional component that corresponds to webpages or subpages. AUnits are single-entry and single-exit, to facilitate structured programming. Each AUnit includes rendering logic which defines its visual appearance in the web browser. AUnits have parent-child relationships to form a tree. This models a website as a hierarchical structure as shown in Figure 1. The leaf level AUnits represent basic components such as HTML forms, which facilitate user interaction. Each AUnit contains a number of *activators* which specify when instances[1] of its child AUnits will be activated at run time. Each activator specifies the child AUnit it corresponds to, the activation condition (a SQL statement) controlling when and how instances of the child AUnit will be activated, the input data for the child AUnit instances, and the operations that are triggered for their outputs. An instance is activated when its activation condition is satisfied in the current state of the application, and deactivated if the condition becomes false.

At run time, the system maintains an *activation tree* consisting of instances of the AUnits activated for each user session. The activation tree at any time represents the part of the application that is currently available to the user through her web browser. When a user performs an operation, such as submitting a form, the leaf AUnit corresponding to this operation returns data to its parent AUnit, which can either perform some operations to update the application state, and/or generate and return data to its parent as well. After the return chain terminates, a new activation tree is constructed based on the updated state. In Hilda, a web application is not a connected graph of individual web pages that allows a user to navigate from a given page to any other page. Instead, the execution of a web application is modeled as a sequence of transitions from one activation tree to another, which continually updates the content shown in the web browsers.

An activator can activate multiple instances of the same child AUnit. The developer defines an *ID* for each instance of an AUnit, which serves as the primary key to distinguish one instance of that AUnit from another. For example, the ID of a *Faculty* AUnit instance is the faculty member's NetID, and the ID of a *FacultyCourse* AUnit instance is the CourseID for the course it corresponds to. The *key* of an instance then is the defined as the set of IDs of its ancestor AUnit instances.

[1]AUnit and AUnit instance is analogous to class and class instance

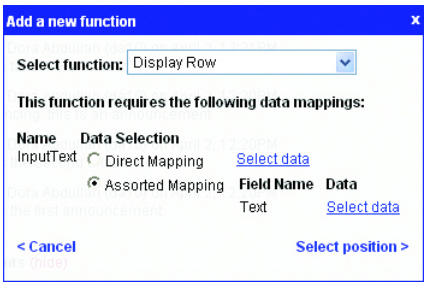

Figure 2: Hilda layer for adding a new function

A Hilda program is executed by the Hilda run time system, which serves as a virtual machine for the language. Hilda is a high level dynamic language which enables us to modify the application logic at run time. The language and its run time system allow us to extend applications in a user-centric manner. In effect, each user of the application can also be a developer. The Hilda run time system supports an authentication model backed by relational databases, which allows only the users having correct access permissions to modify a given part of any AUnit. It prevents users from modifying data and application logic which they do not have access to.

Hilda also provides presentation constructs based on Javascript and HTML, and its run time system provides a GUI interface to dynamically modify both the application logic and the presentation. This enables users to make personalized extensions to an application developed in Hilda, without having to write any Hilda code.

3. DEMO FEATURES

We will demonstrate extensibility in Hilda using a Course Management System (CMS) [2]. We will use the CMS to demonstrate how a user can add various features to the application.

3.1 Single-User Extensbility

We will first demonstrate how a single power user can extend an application — we assume that every user use the same version of the application. The changes made by one user will be visible to others.

3.1.1 Adding New Functions

The Hilda GUI allows the user to extend the functionality of an application by adding new activators or editing existing activators. Recall that an activator specifies the activation condition, input data and output handlers for an AUnit. Hilda allows the user to achieve this on the fly simply by selecting the operand data, and graphically specifying predicates and aggregates on this data. For more complicated queries on the data, Hilda allows users to enter SQL queries.

The user begins by entering the editing mode by clicking on the Hilda button at a corner of the page. She then selects the AUnit corresponding to the function she wants to perform. Figure 2 shows how this is done. Hilda provides some *Basic AUnits*, such as ShowText and InputText, which form the basic building blocks for other AUnits. In Hilda nomenclature, adding a new function on existing data

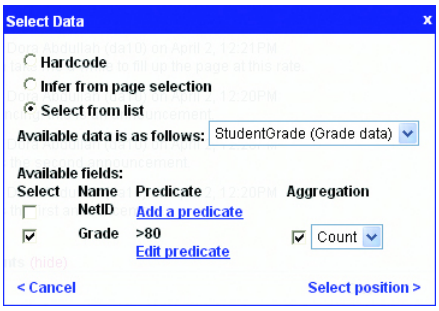

Figure 3: Hilda layer for selecting data

is equivalent to adding a new activator. The selected AUnit may require some input data. If so, the user next selects a mapping from various existing data sources to the AUnit's input relations. The user can either hardcode these relations, select some data from the screen, or select data from a list of available data sources. The user then specifies various predicates and aggregates on the selected data (Figure 3). Finally, the user selects the position in the current module where the new function should appear, and defines its appearance characteristics.

We show how a faculty member, without any background programming knowledge, can integrate the histogram function into the module *FacultyCourse*. The histogram function displays the number of students having grade between different possible ranges, such as 90-100, 80-89, and so on. Each histogram bucket is displayed as the count of the number of students falling in that range. Therefore, a basic module that the user can select to display the histogram is *ShowText*. ShowText requires as input a relation with the field *Text*. Therefore, the user then selects the data for this relation, which comes from the relation StudentGrade. For an example, refer to Figure 3. Each possible bucket of the histogram corresponds to a different instance of ShowText, with different predicates on the StudentGrade relation. The user also specifies the aggregate function on the data, which in this case is a `count` operation. Finally, the user decides where on the current page would she like the histogram to be displayed.

3.1.2 Adding New Data or Modules

We next demonstrate how a user can add new AUnits to the application, which may require creating new relations to store the data. We again consider the histogram function. Suppose the user wanted to group all the instances of ShowText for the histogram, so that the histogram function becomes generic, and can be used anywhere else. Such a generic group of function calls is basically a group of activators, and therefore, an AUnit. Note that a new AUnit can be integrated into any existing AUnit by adding an activator corresponding to it. Hilda also allows the user to create new persistent relations, such as in the case when she wants to add a new AUnit requiring some independent data.

The user adds a new AUnit by clicking on the *New Module* tool button, which asks her to define the input expected by the new AUnit, and the output it might generate. The user then adds activators to this AUnit. After a new AUnit has been created, the AUnit itself appears as an available function in other contexts, and thus can be activated from elsewhere.

3.2 Personalized Extensibility

In the second part of the demonstration, we treat the case when there are different users in the system, and changes made by one user should not affect other users of the system. Hilda achieves this by treating each instance of an AUnit independently. This ensures that changes made to one instance of an AUnit do not affect other instances.

3.2.1 Representing Personalized Extensions

Recall that the key of an AUnit instance is the set of IDs of its ancestor AUnit instances. This key, along with the information about who the current user is ($userid$), defines the operation characteristics of an AUnit, that is, the set of activators in that AUnit which should be included in the version corresponding to that particular user. Hilda maintains exactly one copy of each AUnit, which is stored in a relational database. For each AUnit, there are individual mappings from $(key, userid)$ to a set of activators. A personalized version of this AUnit is then a view over this relational database. Therefore, at the logical level, Hilda actually generates multiple copies of the application on the fly, based on who the user is.

3.2.2 GUI Supporting Personalization

Given that a user can extend AUnits that influence other users, there is a possibility that she cannot navigate to these AUnit by the normal application flow. For example, a faculty member wanting to extend the StudentCourse AUnit, which is part of the *student* view of the course, cannot actually navigate to this AUnit. In the editing mode, Hilda provides an interface that allows a user to jump from one application state to another. This *switch module* feature lets a user specify which instances of a given AUnit would she like to edit, and automatically presents an intersection of the activators common to all of these instances.

3.3 Hilda User Frontend

Finally, we will demonstrate the Hilda GUI and frontend, which allows the user to visually specify declarative queries without any prior knowledge of the database management system. The user-friendly interactive interface provides a WYSIWYG editor for the web AUnits, and facilitates development of more complicated features even for naive users.

4. REFERENCES

[1] G. Booch et al. "The Unified Modeling Language User Guide,The Addison-Wesley Object Technology Series" *Addison Wesley*, 1998

[2] C. Botev et al. "Supporting Workflow in a Course Management System", *Proc. SIGCSE*, 2005

[3] F. Yang et al. "Hilda: A High-Level Language for Data-Driven Web Applications", *Proc. ICDE*, 2006