

Statement of Research, Spring 2007:
**A Next Generation Development Platform
For Data Driven Web Applications**

Fan Yang (yangf@cs.cornell.edu)
Cornell University

Web 2.0 Applications

The success of Web 2.0 applications changes the way how the Web is utilized. Instead of just a convenient way to publish information, in Web 2.0 frameworks, webpages are transformed into Web Computing Platforms with web services as basic building blocks. Like many software vendors, Web site owners are encouraging advanced users to make use of their data and services to create add-on products and more user-tailored services. Users have greater flexibility and control over the tailored applications and services and play very active roles in creating new content and functionalities.

User-Centric Extensions for Web 2.0

We have the vision to empower users to personalize, extend and/or assemble new web applications from existing applications by themselves in Web2.0. Today personalization means that users can change the visual appearance of the application and/or are provided personalized content for their webpages in a predefined way. In other words, personalization is only enabled by the developers who have anticipated the possible ways that users may want to personalize their applications. Therefore developer-centric extension is limited by the clairvoyance of the application designers.

Example of User-Centric Extensions

At Cornell University, we developed an online course management for faculty and students to manage their courses. In the system, faculty can manage announcements, assignments, exams and grading for their courses and students can manage their grades and study groups for assignments. As the system becomes more popular, users demand new functionalities. For example, an instructor wanted students to demo their final course projects and thus needed a way to schedule their demo on line. The functionality of the new component is that the instructor can propose candidate time slots for presentation and students can sign up for the time slot on a first come first served basis. Today, to add the new functionality, developers have to make changes to the application themselves. In our research, we would allow users to easily extend the system themselves, either by building the subsystem from scratch, or by assembling from existing components. If the new functionality turns out to be useful for other courses, other instructors can import this functionality into their course pages as well. By doing so, we allow users to contribute to the community and harness the collective intelligence of all users.

To enable user-centric extension is very challenging. For example, to extend the system with scheduling functionality, we have to change the logic of all tiers in the application e.g. new tables need to be created to store the time slots and sign up information, time slots should be viewable on instructor and students' page, sign up interface should be presented to eligible students etc. Unfortunately, no current systems can allow such user-centric extension.

Research Goal

The goal of my research is to fill the gap between developer-centric and user-centric application extension by developing a platform which allows user to easily personalize and extend the existing application logic. In our platform, users can create value add-ons for their system and to share them among other users in the community. In another word, users would run the web site as if it were their own private copy of the application and they are free to make modifications as coders make changes to an open source desktop program. To make an extension, users are presented graphically a high level and understandable model which captures the application logic of the system and hide the implementation details. Users specify their extension in a graphical way and the new components created is saved and managed by the platform. The evaluation engine will retrieve their personalized logic and translate the logic to either script at client,

executable code at server or external web services invocations (Figure 1). There are many research issues we need to address in our approach:

- We need an abstract model which can capture the logic for all tiers in the application stack. A normal user shouldn't need to understand the details of three tier systems, DHTML, javascript or web services to specify the application logic.
- Users should be presented the abstract model in a reasonable way, so they can understand and create their own business logic in the browser graphically. Users should also be able to use and extend the functionality created by others if permitted.
- The system should apply user-defined business logic efficiently. We shouldn't expect users to take care of performance optimization details e.g. partition the logic across server and client to optimize performance etc.

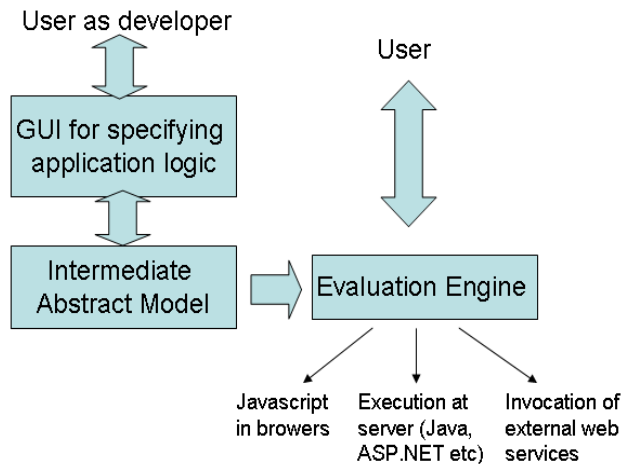


Figure 1, System Architecture

System Design Details

Graphical Interface for Extension: To extend application logic with our platform, normal users will view graphical representation of the construct units in our abstraction model and specify the logic graphically while advanced users can write code directly in the modeling language. The interactive logic for the system will be specified similar as in event based programming which is nature for normal users to understand. The GUI will then generate the intermediate representation of the application logic in our abstraction model which will be executed by the evaluation engine.

Abstract Model: In the HILDA project at Cornell University, we developed a high-level declarative programming language. It provided a unified programming model to express application logics for back end database, application server and front end user interactions.

Our abstraction model of the application logic will be based on HILDA and we will modify HILDA's inheritance mechanism to model the extension of systems made by users. In the new model, users can append and/or override the existing features of a component and wouldn't affect the original component developed and used by others. The traditional object oriented inheritance model will not satisfy our need though. We need the power to express the extension a sub system which may contains multiple components that interact with each other. We would adapt the idea from the package inheritance which is more powerful than class inheritance for extensibility.

Evaluation Engine: Our evaluation engine will be based on the runtime system developed for HILDA language. The original runtime system was built upon the standard three tier architecture and evaluates the business logic specified in the HILDA model. It provides a virtual machine for the model and hides details of implementation and cross layer optimizations. The system partitions the logic automatically across client and server and caches the necessary data at the client side to optimize users' respond time.

We would extend the runtime system so that the application logic can be partitioned into parts that run in the browsers, at the server and invocations of web services provided by external sources. We will build the support for managing different versions of the components extended by different users into the platform. We would study the method to store and share users' extension efficiently. More optimization goals will also be integrated and managed by the platform e.g. scalability. What's more, access control policy is needed for users to enforce the restriction on whom they want to share the extension with and what part of the system they are allowed to extend.