WebCity: A WWW-based Hypermedia Environment for Software Development

Wenyu Jiang, Gail E. Kaiser, Jack Jingshuang Yang, Stephen E. Dossick
Department of Computer Science, Columbia University
1214 Amsterdam Avenue, Mail Code 0401, New York, NY 10027

Abstract:

Many SDEs (Software Development Environments) have been built to help increase software productivity. However, these systems often have several deficiencies: inability to operate/act on external data; hardwired behaviors; and finally lack of a universal remote access mechanism for geographically distributed users.

In this paper we first introduce *OzWeb* , a system which combines the elements of SDE, Workflow Management, and the World Wide Web (WWW). We explain how it provides a solution for the above three problems. Then we describe *WebCity*, an instance of SDE which applies *OzWeb* 's technology to a real environment. We will discuss how it solves those problems with SDE, and our experience in building and using it. Finally we compare *WebCity* with similar systems that also deploy Web technology.

1 Problems

Various Software Development Environments (SDEs) have been available and in use for many years. They range from personal software packages such as Borland C to multi-user environments (e.g, Sun NSE) providing configuration management, concurrency control, etc.

However, there are several weaknesses in these systems: first, they are usually self-contained and rarely interact with external data. For instance, Borland C provides useful on-line help, but when a user finds an on-line reference for a toolkit he used in his project, he cannot add it to Borland C and integrate it with the existing help information. Therefore, the user has to store and search the reference separately, defeating the purpose of an integrated design environment. Storing the external references separately also makes it easy to be forgotten or neglected, a common problem during change of personnel involved in a large software project.

Second, the functionality and behaviors of SDEs like Borland C are very much hard wired, making it difficult to be customized for any projects with special demands.

Finally, most SDEs don't have a consistent interface for remote access, a drawback for geographically distributed users or those working at home. Software such as Lotus Notes provides users with automated download/upload of documents and programs, but it uses a proprietary communication protocol, making it difficult to be accessed over the Internet, the largest network in the world.

2 Approaches

Many research SDEs (e.g, *Oz* [1]) solves the second problem by using a general workflow engine. A workflow modeling language is used to control the engine's execution.

The Web provides an infrastructure for solving the remaining problems, through the use of hyper links (to refer to external data) and its ubiquitous availability (to facilitate remote access).

Apparently an appropriate combination of SDE, the Web, and workflow management can provide a promising solution for these problems. Therefore, we developed the *OzWeb* [2] system, based upon which we built *WebCity*: a WWW-based hypermedia code development environment.

3 OzWeb

Oz [1] is an SDE system the authors developed at Columbia University, aimed to provide flexible workflow automation with a rule-based workflow engine.

To solve the remaining two problems, we extended *Oz* to support HTTP at the front end. It responds to a web browser by returning an HTML page with links of data objects and rules, the user browses data and executes rules by clicking on appropriate links. We also added special code to handle read/write of web documents (URLs) at the back end, by introducing a special object type called WebObj to the system.

With an HTTP frontend, the user can access the environment via any web browser from virtually anywhere. With back end handler for web documents, the user can store references to external data on the web easily in the environment. It can be even used to perform workflow on web documents, e.g, whenever a document was detected to have changed. The modified version of *Oz* , owing to its Web-aware nature, is then named *OzWeb* .

OzWeb does more than storing URLs. A WebObj object can have 3 different types: Reference, Copy and Updatable. "Reference" is the default, it means whenever *OzWeb* performs a read operation, it should get the most recent data from the Web. "Copy" means once a local cache file is present, the cache should always be used, possibly useful for mirroring. "Updatable" means the user has permission to update the document represented by its URL via HTTP PUT.

OzWeb consists of several components: the workflow engine uses specified workflow model (rules) to control its automation, the object-oriented database (objectbase for short) stores program data and information persistently, and the transaction manager provides crash recovery and concurrency control. Finally, a shell script is used to wrap up external tool (e.g, gcc). The server

is responsible for invoking the tool. All three components are highly customizable, which makes it highly convenient for building SDEs with arbitrary demands.

4 WebCity

OzWeb is a general-purpose server program, therefore it must run on a specific environment, which defines its own workflow model, data schema, and concurrency policy. To verify the applicability of *OzWeb*, we built *WebCity*, the environment we use to develop our own research project.

Most of its features are normal software development activities, including build, compile, edit, check in, check out files, support of local workspaces and local projects, etc. All of these activities are implemented by rules, since *OzWeb* uses a rule-based workflow engine.

WebCity organizes program data as a tree of objects, there is a master project tree, serving as the main repository, each user has his own local workspace, under which contains multiple local projects, with source files checked out from the main repository.

In order to test out the usage of web documents, we added several features to *WebCity*:

First, every source file (CFILE, HFILE, etc.) is also a subclass of WebObj, with its URL representing a HTMLized version of the source code, which we dubbed HiC'ed code, because it is automatically generated by a homegrown utility called HiC. In HiC'ed code, every function call or data type usage appears as a hyper link, pointing to its definition. This provides convenient traversal of source code without having to run a debugger.

Second, every local project, local workspace, or even the entire project can have a DOCUMENTATION child object, which can further have any number of WebObj child objects. Therefore, we can easily create references to external web documents at different granularity: documents pertinent to a local project, to a local workspace, or to the entire project.

We have used *WebCity* for several months, its HiC facility proves to be very useful, and suits well for use on the web. Flexibility of adding external web documents helps us organizing information in an orderly fashion. Finally, the users can work either at home or in the office conveniently, without having to worry about file download/upload.

Most of the advantages presented here are attributed to *OzWeb* , and not specific for *WebCity*, since *OzWeb* serves as the foundation for the web frontend user interface and backend data operation. But through use of *WebCity* we see applicability of *OzWeb* .

As a side statistics, *WebCity* has about 80 rules, 44 classes, 30 shell scripts (for tool wrapping), it's a reasonably complex software development system.

5 Extensibility of *OzWeb* and *WebCity*

Here are some interesting scenarios on the usage of WebObj types that we didn't have time to try, but we claim they can be done with relative ease:

- Reference: we are using an on-line manual of a beta-version toolkit. To keep the user up to date with the toolkit info, we can define the "build" rule such that if the manual object's timestamp has changed since the last project rebuild, notify the user.

- Copy: we found an on-line latex manual and don't want to risk losing it, then specify the WebObj object's type as Copy.

- Updatable: Suppose you put the source code and help manual of a project of *WebCity* on web site A, for which you have PUT permission. To keep download users up to date, you can declare the help manual object of type Updatable, then define the "build" rule to let it automatically "trigger" another rule to update web site A via HTTP PUT.

Although *WebCity* currently supports only C, rules and schemas can be extended to support another programming language such as Java. Similarly, it can be extended to support tasks like design and testing as well as coding. All are due to the flexibility of *OzWeb* , especially that of the workflow engines and objectbase.

Of course, a general workflow engine despite its flexibility, is usually not easy to use. Writing a workflow model is not trivial, especially when there are many tasks interacting with each other.

6 Comparisons

- Lotus Notes supports simple workflows and document routing, replication, etc. However, its workflow engine is too simple to be used in a software development context.

- *WebMake* [4] is an architecture designed to support distributed software development over the Web. It is based on the CGI (Common Gateway Interface). When the user issues the command to "make" a given software component, a CGI program is invoked to process the request on the server where the code resides. It evaluates code dependencies, and sends further requests if the code to be compiled depends on code residing on another machine. *WebMake* is useful where *Make* fits best. However, it only supports batch jobs (e.g, recompilation), and therefore cannot be easily applied to interactive tasks.

- IDEs (Integrated Design Environments) like Borland C have all its facilities seamlessly integrated, providing users with a covenient and consistent user interface. In contrast, *WebCity* and many research SDEs provide only loose tool integration, usually by wrapping tools with shell scripts.It is a trade off between flexibility and better user interface.

7 Related Works

WEBDAV [5] is an Internet draft for distributed authoring and versioning over the Web. It can provide a foundation for distributed workflow on the Web.

WebWork [6] is a Workflow System with a web front-end for building customized workflow models. It aims at how to create a workflow model rather than what functionality to provide in a workflow modeling language.

References

[1] Israel Z. Ben-Shaul and Gail E. Kaiser: *A Paradigm for Decentralized Process Modeling* 1995 Kluwer Academic Publishers, Boston MA.

[2] Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang and Jack Yang: *An Architecture for WWW-based Hypercode Environments* 1997 International Conference on Software Engineering.

[3] Stephen E. Dossick and Gail E. Kaiser: *WWW Access to Legacy Client/Server Applications* 1996 5th International Conference on the World Wide Web.

[4] Michael Baentsch, Georg Molter and Peter Sturm: *WebMake: Integrating distributed software development in a structure-enhanced Web* WWW95, Computer Science Department, University of Kaiserslautern, Germany.

[5] WEBDAV Working Group: Y. Goland, E. J. Whitehead, Asad Faizi, Stephen R. Carter and Del Jensen: *Extensions for Distributed Authoring and Versioning on the World Wide Web – WEBDAV* 1997 Internet Draft.

[6] John A. Miller, Devanand Palaniswami, Amit P. Sheth, Krys J. Kochut and Harvinder Singh: *WebWork: METEOR$_2$'s Web-based Workflow Management System* 1997 Department of Computer Science, University of Georgia.