A World-Wide-Web Architecture for Collaborative Software Design

T.C. Nicholas Graham
Hugh D. Stewart
A. Reza Kopaee
Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada
graham@cs.queensu.ca

Arthur G. Ryman
IBM Toronto Laboratory
1150 Eglinton Avenue East
Toronto, Ontario, Canada
ryman@ca.ibm.com

Rittu Rasouli
Department of Computer Science
York University
4700 Keele Street
Toronto, Ontario, Canada
rasouli@cs.yorku.ca

Abstract

Rosetta is a tool that supports the creation of object-oriented design documents, and automatically checks the conformance of Java implementations to those designs. Rosetta is based on a novel WWW architecture, supporting collaborative use with heterogeneous development tools under a coevolutionary development process. Rosetta has been used extensively in our research group and in teaching at the first year university level, and is currently undergoing industrial field trials. Rosetta has proved successful, but its deployment over the WWW has not proved as transparent to users as we had hoped.

1. Introduction

In order to collaborate as teams, software engineers need to be able to communicate the design of their software to other team members who may be physically separated from them, and to future maintenance programmers who may be working after the original development team has disbanded. Automated tools for design have the potential to support communication by allowing the electronic sharing of designs. However, poor support for communication among developers is cited as one the primary reasons why Computer-Aided Software Engineering (CASE) tools are not widely used [8,9].

In order to support collaborative use, a design tool must provide:

- *Easy creation and sharing of design documents,* where members of a design team can immediately access documents created by their colleagues;
- *Flexible communication,* where the development processes enforced by the tool do not interfere with developers' formal and informal communication [10];
- *Flexible deployment,* where developers within a team are allowed to use different code development tools, perhaps running on different operating systems.

This paper presents Rosetta, a light-weight tool for object-oriented design using the Unified Modeling Language (UML) [15] notation. Rosetta meets the three requirements outlined above by using a novel World Wide Web architecture. Design documents are written in HTML, and may contain embedded UML diagrams. These diagrams are stored on a server, allowing them to be viewed and edited from anywhere on the Internet. A checker tool tests the conformance of code implementations to these design documents. Rosetta is therefore appropriate for a coevolutionary development process [4], since design and code can be created in any order. The checker tool can be used at any time to establish how closely the design and code match. As well, the checker tool permits maintenance programmers to assess the trustworthiness of legacy designs. Rosetta is compatible with a wide range of code development tools: since code is produced independently of designs, the design tool does not interfere with the coding process. A novel Java servlet-based [7] architecture permits the conformance checking of code that is stored in files or in a distributed repository.

Rosetta has been implemented. It has been used in the development of two significant projects in our research lab, in teaching object-oriented design to first year university students, and is currently undergoing industrial field trials.

Rosetta's World Wide Web architecture has proved to be a successful mechanism for deploying a software design tool. We have, however, experienced difficulties creating a fully transparent web-based installation process.

This paper is organized as follows. Section 2 gives an example of the style of design document that can be created

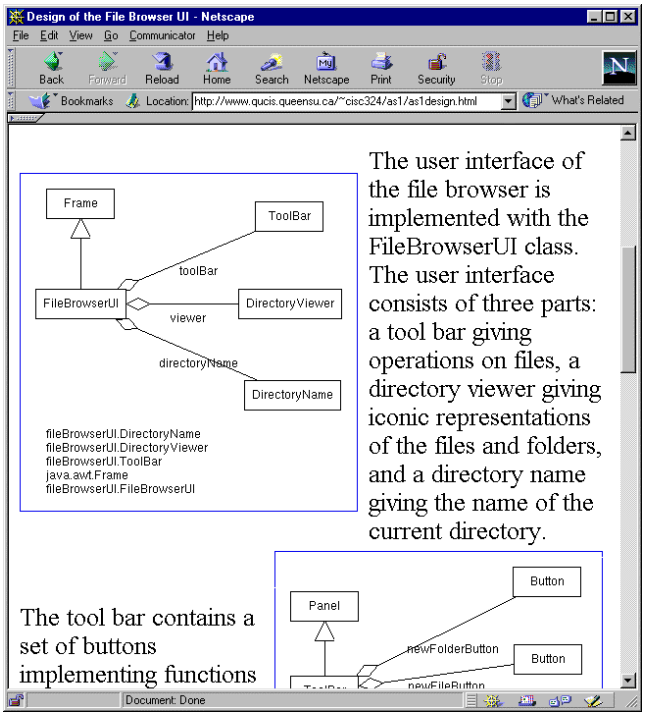

Figure 1: Example Rosetta design document. Designs consist of text and diagrams in the UML [15] notation. Nodes in diagrams may be hyperlinked to arbitrary documents, such as other designs or *javadoc* [11] documentation. Design documents are created in HTML, and can be viewed in standard browsers.

with Rosetta. It then introduces the Rosetta architecture, and explains how the architecture supports collaborative software development. Section 3 shows how Rosetta's WWW architecture provides flexibility with respect to process, allowing developers to adapt the tool to their existing collaborative work practices. Section 4 shows how Rosetta's WWW architecture provides flexibility with respect to toolset. Finally, section 5 evaluates the tool's use in practice and discusses our experiences with developing applications for the World Wide Web.

2. Collaborative Creation and Viewing of Design Documents

In a survey of professional software developers, Kraut and Streeter identified that of eighteen ways development teams coordinate their activities, developers cite *discussion with their peers* as the most important [10]. Following a study of 55,000 hours of development activity, Norcio and Chmura determined that discussion among software engineers is correlated with progress in design [12]. Such communication becomes much more difficult, however, when development teams are distributed [16].

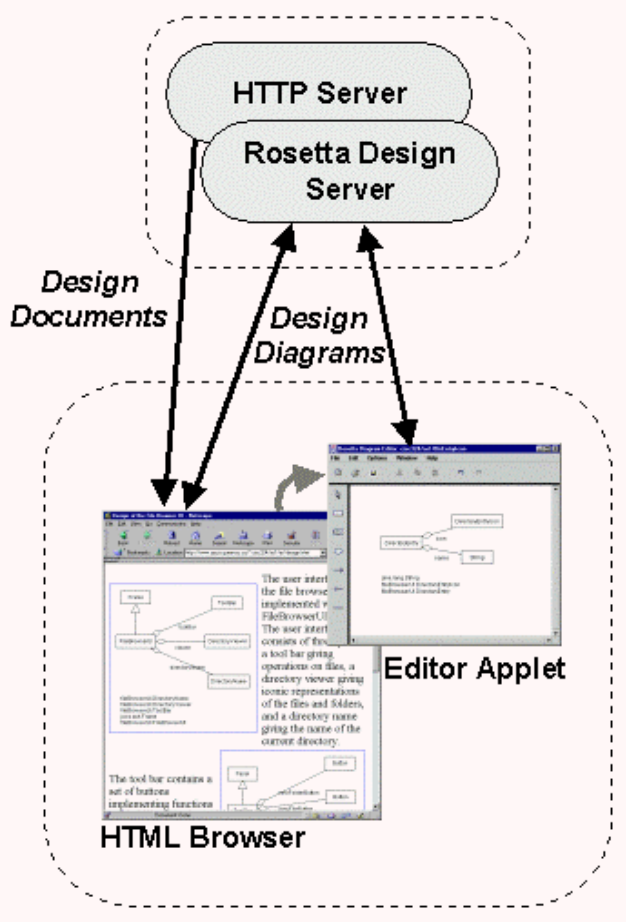

Figure 2: Architecture for viewing and creating design documents. Documents are standard HTML, served by any HTTP server. Diagrams are embedded within the documents, and served from the Rosetta server. An editor applet allows designers to create and modify diagrams from anywhere on the Internet.

Software design tools seem to promise support for communication between team members. Once they are created in electronic format, designs can easily be shared with other developers, and can form a basis for communication. According to Jarzabek and Huang, however, a lack of flexible support for informal communication is an important reason why current CASE tools are not widely used [9].

In order to support the work of distributed development groups, a software design tool must support the remote creation, editing, and viewing of design documents. The tool must have minimal barriers to use: developers should not have to install new software in order to view colleagues' designs, and should not have to learn complex new tools to modify or update those designs. As well, the design tool must provide access and concurrency control.

In order to support the most flexible distributed viewing of design documents, Rosetta design documents are written

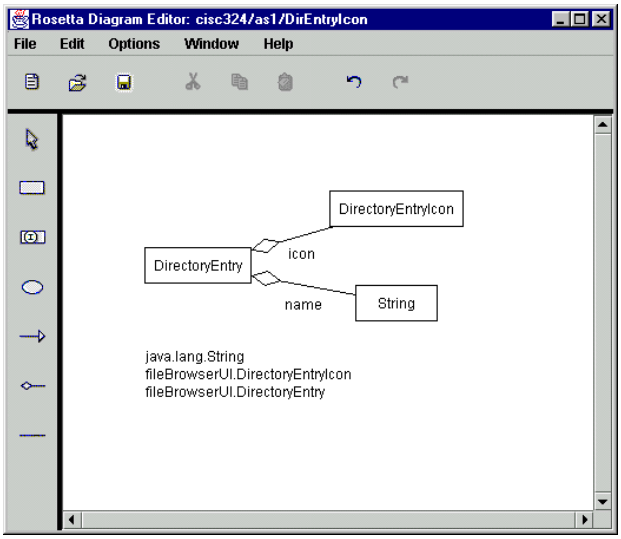

Figure 3: The Rosetta Editor supports the creation of UML [15] class and object diagrams. Diagrams are stored in a central repository, allowing them to be edited from any location on the Internet.

in HTML, using any HTML editor. Design documents are comprised of text and class diagrams in the UML notation. Since they are developed using standard HTML, design documents can be viewed by anyone with access to a web browser, from any location on the Internet. Figure 1 shows a design document created using Rosetta.

A document may contain any number of class diagrams. Class diagrams consist of classes, interfaces, and primitive types that may be related via general association, aggregation and inheritance. These diagram elements may be hyperlinked to related information. By default, class and interface nodes are hyperlinked to the *javadoc* [11] documentation describing their interfaces.

2.1. Document Server Architecture

To support collaborative work, it must be possible to create, edit and view design documents from distributed sites. It must also be possible for groups of people to concurrently create and modify designs.

The World Wide Web architecture shown in Figure 2 meets these requirements. Designs are HTML documents, served by an HTTP server. Design diagrams are embedded within these HTML documents using a JavaScript [5] tag. Each JavaScript tag loads a diagram from the *Rosetta Design Server*. The design server serves each diagram as a GIF image with an associated image map, which allows each element in the diagram to be hyperlinked.

The design server also permits diagrams to be loaded and saved in a structural form suitable for editing, and provides access control enforcing diagram ownership and concurrency control. This allows diagrams to be safely viewed and edited from any location on the Internet.

Since diagrams are stored on a central server, they are available for editing by anyone who has write permission for the diagram, from any location. Group work is fully supported, since any number of people can edit the design diagrams (although only one person at a time can hold the write lock to a given diagram). As well, by clicking the reload button of their HTML browser, updated diagrams are immediately available to anyone viewing the design documents.

The central design server is a Java application, and can therefore run on any machine and operating system for which there is a Java Virtual Machine.

2.2. Creating Design Documents

A major goal of Rosetta was to reduce barriers to adoption by making the creation of design diagrams simple, and by requiring no special tools or installation. The WWW architecture described in the previous section was designed to meet these goals. Section 5 evaluates the success of the architecture.

While HTML design documents can be created using any HTML editor (the example documents in this paper were prepared using Netscape Composer), designers create the UML diagrams embedded in those documents using the Rosetta editor, shown in figure 3. This editor is a full-featured editor for UML class and object diagrams. It includes support for multiple windows, copy & paste and undo operations, the attachment of URL links to diagram elements, and font selection. The Rosetta editor is implemented as a Java applet. The editor therefore requires no installation; it is simply invoked from a web page. As an applet, the editor runs on a wide range of machines and operating systems.

Diagram *open* and *save* operations are performed by connecting with the design server. The open operation locks the diagram to avoid concurrent editing of the diagram by multiple designers. The save operation saves the diagram structure. When a diagram is saved, the editor also creates a GIF image of the diagram and an associated image map. The image map describes links from the diagram elements to other documents. On saving a diagram, the editor also writes a JavaScript tag referring to the diagram to the web browser's Java console. To insert the diagram into a design document, the designer simply pastes this tag into his/her HTML design document.

In summary, Rosetta provides for the easy creation, editing and viewing of design documents using standard web browsers. This allows development teams to share design documents across the Internet. A central design server stores the design diagrams. Since the diagram editor is an applet, designers do not need to install special software in order to edit designs.

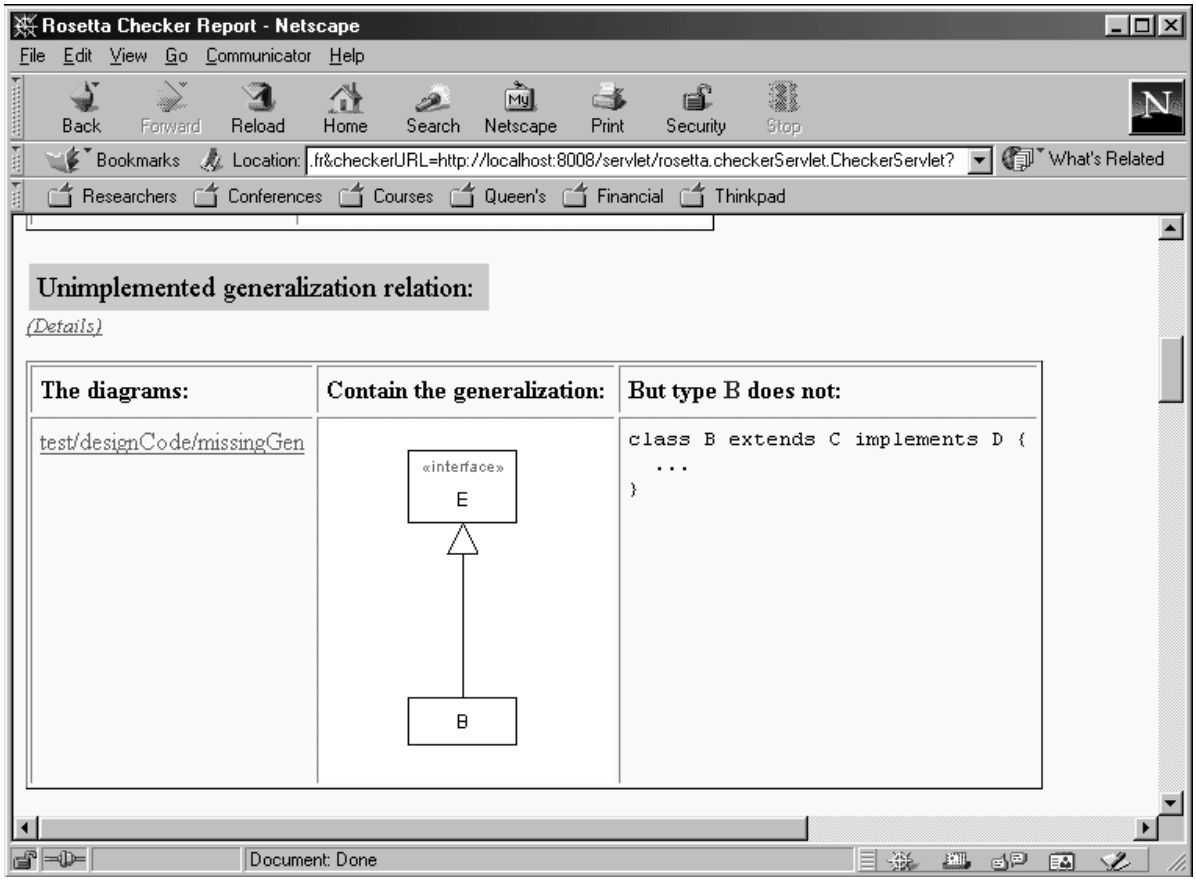

Figure 4: An example checker report. The checker identifies inconsistencies among design diagrams and inconsistencies between design and code. Checker reports are HTML documents.

3. Flexible Communication through Flexible Process

Empirical studies show that interaction between members of a software engineering team takes many forms, ranging from informal to highly structured [10,16]. Design tools should promote this interaction by allowing developers to communicate in flexible ways [9]. The design tool should not enforce a process that interferes with this communication.

For example, imagine that a developer wishes to propose an alternative design for part of a system whose implementation is already underway. A design diagram for this new proposal may be inconsistent with the rest of the design, and may be inconsistent with already implemented code. The design tool should not interfere by insisting that these consistency issues be resolved before the design can be updated.

Existing software design tools [6,14] tightly couple design and code so that they stay synchronized. This approach has the benefit of ensuring that the design accurately reflects code, but has the cost of enforcing work practices that may be inappropriate in a distributed setting. Tools supporting the ViewPoints approach [2] and coevolutionary design [1,4] recognize that for distributed design, it is necessary to tolerate inconsistency.

As a project proceeds, it is often the case that design documents cease to reflect the system as implemented [3]. In order to make informed use of the information contained in design documents, software engineers need to able to easily assess the accuracy of that information. That is, they need to be able to easily determine whether the implementation conforms to the design.

3.1. Conformance Checking

While Rosetta allows design and implementation to coevolve with arbitrary inconsistency, it also allows developers to easily discover where those inconsistencies exist. In Rosetta, there is no direct link between design and code. Instead, designs are created (as described above in section 2.2) independently of code development. At any time, developers can invoke a design-vs.-code conformance checker to test how closely the design matches the code. The use of a confromance checker allows code and design to evolve independently, while making it easy to test where the two differ.

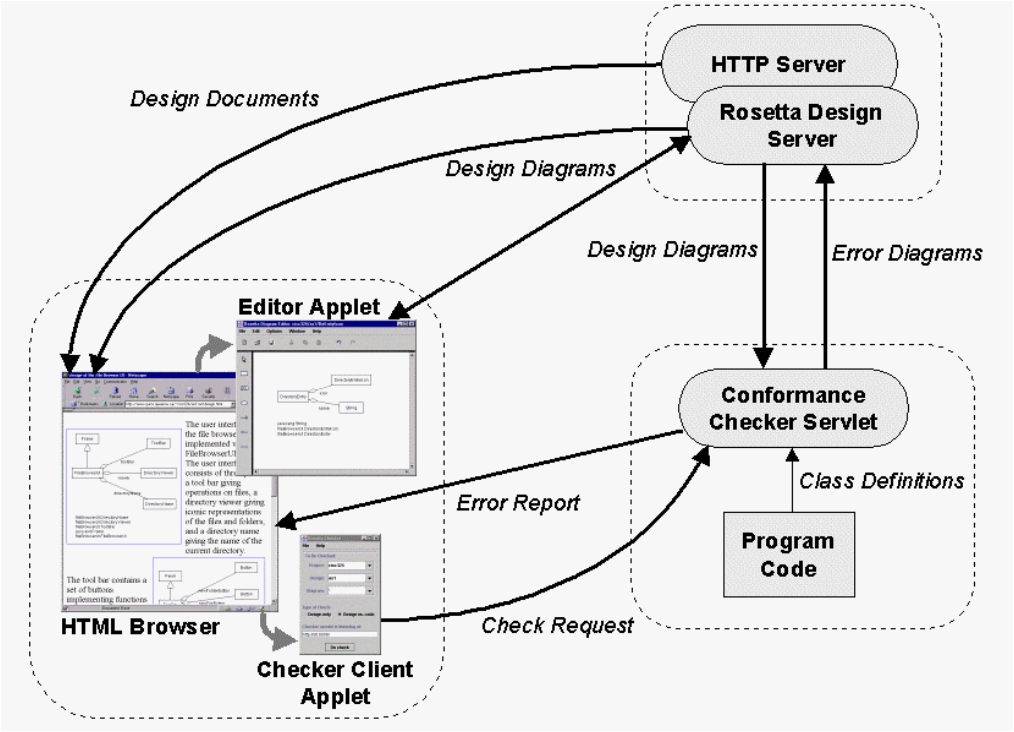

Figure 5: Complete architecture of Rosetta. Each developer uses their browser to view and edit designs. Developers invoke check requests through a checker applet. A checker server performs conformance checks, and returns the result as an HTML error report.

Conformance checking is based on a simple set of rules that specify how designs should be implemented in code. Wherever these rules are not satisfied, a warning message is generated specifying how the code and design differ. Figure 4 shows the form of these warnings messages in a report from the checker. Specifically, the rules are:

- UML classes/interfaces map to Java classes/interfaces.
- UML aggregation relations map to Java fields or accessor methods.
- UML inheritance relations map to Java *extends* or *implements* clauses.

The problem of checking UML multiplicities is in general uncomputable. Multiplicities are partially checked by requiring the target multiplicity "*many*" be mapped to container objects, such as array or vector types.

The conformance checker allows developers to decide what inconsistencies between design and code are tolerable, and what needs to be fixed immediately. This approach supports coevolutionary development, where design and implementation proceed in parallel, potentially become inconsistent, and synchronize at times the developers consider appropriate.

Conformance checking also helps maintenance programmers who must assess the accuracy of designs, often when the designers are no longer available. The conformance checker automatically reveals where the code has evolved since the last update of the design documents.

In summary, Rosetta does not enforce a design process that might inhibit some forms of communication among developers. Designers are free to develop designs and code in any order, and are free to manage inconsistency as they choose.

4. Flexible Deployment

Design tools for team-based development must also be flexible with respect to development tools. The developers within a team will not necessarily use the same code development tools, particularly when development teams span companies or company sites. It is unrealistic to expect that developers will change development tools to suit the requirements of a design tool, or that the eventual maintainers of the code will use the same environment as the developers. Therefore, it is important that a design tool not be bound to a particular code development tool, but that

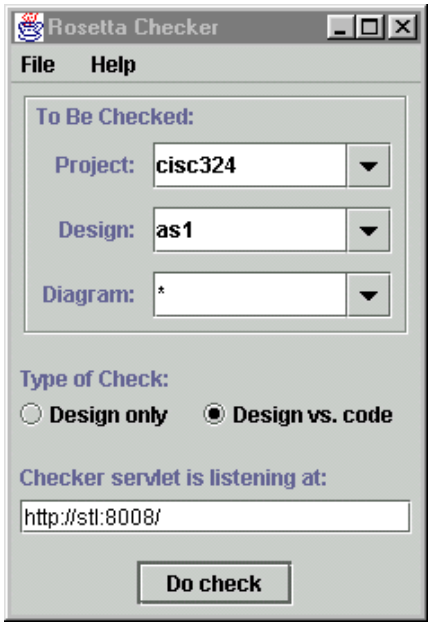

Figure 6: The checker client applet. Users specify what diagrams to check, and whether to check the internal consistency of the design, or the conformance of code to the design. The URL of the checker server can be specified.

the design tool instead be usable within a heterogeneous development environment.

As well, to be successfully used by a group, all members of the design team must be willing to use the design tool. The tool should therefore be easy to install and easy to use.

The Rosetta approach of loosely coupling design and code through conformance checking provides the basis for supporting heterogeneous code development tools. As described in section 2, editing and viewing designs does not impact the coding process, and is thus independent of code development environments. The code and design are linked only by the conformance checker, which must have access to both. As described in section 4.1, below, the implementation of the checker as a Java servlet allows the same checker to be used with a wide range of code development tools on a variety of operating systems.

The use of a WWW architecture simplifies Rosetta's installation process. Since the tools are implemented as applets, they can be run in a web browser simply by going to the appropriate web page. However, as described in section 5, this applet deployment did not prove as simple as we had hoped.

4.1. Architecture for Conformance Checking

Figure 5 shows the architecture of the complete Rosetta tool. Developers use the conformance checker via the *checker client applet* shown in Figure 6. This applet allows users to specify what parts of the design are to be checked (in Figure 6, the user has specified that all diagrams in the *as1* design of the *cisc324* project are to be checked). Users may specify an internal consistency check amongst all the diagrams in a design, or a design-vs.-code conformance check.

The checker client asks the checker servlet to perform the check. The checker servlet replies with an HTML check report like that of Figure 4. The check report is displayed in the user's browser.

The checker servlet reads design information from the design server. To obtain the corresponding code, the checker servlet must be run on a machine where it has access to that code. When it generates check reports, new diagrams illustrating inconsistencies may be written back to the design server. The checker servlet runs inside a minimal HTTP server. This means that check requests are HTTP requests, and that the response is an HTML check report suitable for display in a browser. The use of HTTP for communication between the checker client applet and the checker servlet allows the checker servlet to be located on any machine, without compromising Java's applet security model.

The checker servlet is compatible with a wide range of code development tools. Most Java development tools store code on the file system. Giving the checker access to that program code is as simple as running the checker on a machine where it has access to the file system. Some tools (such as IBM's VisualAge for Java product) store code in a proprietary repository to support distributed code development. For such environments, the checker can be run within the development environment itself to gain access to the repository.

To demonstrate the flexibility of this architecture, we have run the checker on code developed in VisualAge for Java, Sun's Java Workshop, and Sun's Java Development Kit, on the Solaris, Linux, Windows 95, and Windows NT operating systems.

While the architecture of Figure 5 is very flexible, Rosetta can in practice be deployed in simpler configurations. The design server is actually a minimal HTTP server, used to serve the editor and checker client applets and the Rosetta on-line help pages. Therefore, if the design documents are stored on the same machine as the design server, there is no need for a separate HTTP server. As delivered, the design server includes a checker servlet. Therefore, if the code can be located on the same machine as the design server and its checker servlet, there is no need to run a separate checker servlet.

In summary, Rosetta loosely couples code and design through a conformance checker. This means that the tools used for creating and viewing designs do not interfere with those used for coding: developers are free to choose any code development environment. The conformance checker requires access to the code, and is thus implemented as a checker servlet located where that code lives. We have shown that the checker can be used without modification

on code produced by a number of development environments.

5. Evaluation

In order to evaluate the success of the Rosetta architecture, we implemented Rosetta in the Java programming language. The tool has undergone the following trials:

- Three developers used Rosetta within two significant projects within the Software Technology Laboratory at Queen's University. While the developers were within our research group, they were not members of the Rosetta project itself, and were using the tool at their own discretion.
- Approximately 90 students at Queen's University are currently using Rosetta as part of a first year introduction to computer science. The tool is being used to document and check modest programming assignments. The use of the tool is voluntary.
- At the time of writing, industrial field trials are commencing with volunteers at the IBM Toronto Lab.

Once successfully installed, the Rosetta tool received a very positive reception. The process of creating designs and of checking designs against code is simple, and in practice quickly learned. We will not be able to definitively conclude that the tool is successful until our field trials have progressed further. However, the enthusiastic use of the tool within our own research group indicates that developers find the tool easy to use and easy to adapt to their own work practices.

5.2. Installing Rosetta

On the other hand, the process of installing Rosetta can be frustrating. While web-based architectures such as the one described in this paper are effective, they do not yet permit a simple, transparent installation process.

We found the Java virtual machines in standard web browsers to be too slow and unreliable to use as execution platforms for Rosetta. As well, the latest browsers typically contain virtual machines several releases behind the current version of Java, and the browsers installed on peoples' machines typically lag several releases behind the latest version of the browser. Additionally, the two most popular browsers do not implement compatible versions of Java. In order to use the Rosetta editor and checker applets, users are therefore often obliged to update their browser. (Fortunately, documents can be viewed with older browsers, as viewing does not involve running Java applets.) Users find the potential requirement of upgrading their browser a significant barrier to using Rosetta.

We have recently switched to using Sun's Java Plug-in [17], which replaces the virtual machines built into browsers. The Java Plug-in works very effectively, delivering the speed of Sun's fastest virtual machine. Furthermore, the Java Plug-in behaves identically across browsers, solving the problems of cross-platform incompatibilities and the general unreliability of browser virtual machines. However, the Java Plug-in brings its own problems. Users are obliged to download the specific version of the Plug-in that is compatible with Rosetta. While most people have a browser installed, the Plug-in is relatively rare. Since the Plug-in works only with more recent browser versions, users often need to both upgrade their browser and download the Plug-in—a combined download exceeding 10 MB. Finally, installing the Plug-in requires administrator privileges on Windows NT and Unix systems, which users may not have.

In summary, the flexible architecture of the Rosetta tool works very well once the tool has been installed. However, users must often perform significant software installation in order to use the tool. In this respect, web-based delivery falls short of its promise of removing the need for software installations.

6. Conclusions

In this paper, we have presented a World Wide Web architecture supporting collaborative software design. We have argued that such an architecture should support the viewing and editing of designs from distributed locations, the flexible work processes that are required in collaborative work, and the ability to operate within an environment of heterogeneous development tools. We have shown that the Rosetta architecture meets these requirements by supporting HTML design documents incorporating diagrams created with an editor applet that talks to a central design server. To support flexible process and heterogeneous tools, the Rosetta architecture decouples design and code, and provides a conformance checker to report on inconsistencies between the design and the code.

We conclude that the technology for deploying software tools over the web is still immature; such deployments often require users to shoulder the burden of downloading and installing new web browsers and web browser plug-ins.

Acknowledgements

The work described in this paper was supported by the IBM Centre for Advanced Studies and by Communications and Information Technology Ontario (CITO). The graphic design of the Rosetta Editor was performed by the IBM Media Design Studio. The design of the editor benefited from the advice of the IBM User Centered Design Lab.

REFERENCES

1. Brown, J., Graham, T.C.N. and Wright, T., The Vista Environment for the Coevolutionary Design of User Interfaces, In Proceedings of CHI'98, ACM Press, 376-383, April 1998.

2. Easterbrook, S., Finkelstein, A., Kramer, J. and Nuseibeh, B. Coordinating Distributed ViewPoints: the Anatomy of a Consistency Check. *International Journal on Concurrent Engineering: Research and Applications*, 2,3, 209-222, 1994.
3. Finnigan, P., Holt, R.C., Kalas, I., Kerr, S., Kontogiannis, K., Muller, H., Mylopoulos, J., Perelgut, S., Stanley, M. and Wong, K. The Software Bookshelf. *IBM Systems Journal*, 36, 4, 564-593, November 1997.
4. Fisher, G., Redmiles, D., Williams, L., Puhr, G.I., Aoki, A. and Nakakoji, K. Beyond Object-Oriented Technology: Where Current Approaches Fall Short. *Human-Computer Interaction*, 10,1, 79-119, 1995.
5. Flanagan, D. *JavaScript: The Definitive Guide*, O'Reilly & Associates, 1998.
6. Hui, Alan, *IBM VisualAge UML Designer: An Integrated Object-oriented Analysis and Design Too.*, http://www.software.ibm.com/ad/smalltalk/about/umlwhite.pdf
7. Hunter, J. and Crawford, W. *Java Servlet Programming*, O'Reilly & Associates, 1998.
8. Iivari, J. Why Are CASE Tools Not Used? *Communications of the ACM*, 39,10, 94-103, Oct.1996.
9. Jarzabek, S. and Huang, R. The Case for User-Centered CASE Tools. *Communications of the ACM*, 41, 8, 93-99, August 1998.
10. Kraut, R.E. and Streeter, L.A. Coordination in Software Development. *Communications of the ACM*, 38, 3, 69-81, March 1995.
11. Niemeyer, P. and Peck, J. *Exploring Java*, O'Reilly & Associates, 1996.
12. Norcio, A.F. and Chmura, L.J. Design Activity in Developing Modules for Complex Software. In Soloway, E. and Iyengar, S., editors, *Empirical Studies of Programmers*, 99-116, Ablex Publishing, 1986.
13. Parnas, D.L. and Clements, P.C., A Rational Design Process: How and Why to Fake it. *IEEE Transactions on Software Engineering* SE-12,2, 251-257, Feb. 1986.
14. Quatrani, Terry, *Visual Modeling with Rational Rose and UML*, Addison-Wesley, 1998.
15. Rumbaugh, J., Jacobson, I. and Booch, G. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.
16. Seaman, C.B. and Basili, V.R. Communication and Organization in Software Development: An Empirical Study. *IBM Systems Journal* 36, 4, 1997.
17. Sun Microsystems, *Java™ Plug-in Overview*, http://www.javasoft.com/products/plugin/1.1.1/.