Abstract
Web services turn a web server into an application server. Suppose the application is Java programming, and specifically, compiler construction, what kind of web services are convenient and what can they be used for?

The talk demonstrates the Web Compiler Service wcs and discusses its architecture. Wcs implements Java-based compilation on a web server and execution in an applet. It supports rapid prototyping, experimentation with Java programs and Java-based compiler generation tools, and specifically, the delivery of very interactive tutorials. The client only needs to execute unsigned Java applets; the software development tools reside on the server.

We have used the technology to implement tutorials on topics in introductory programming as well as language recognition and interpretation. It can also be employed to enable experiments with Java-related tools without the need to install them first.

Introduction
Scanner and parser generators such as lex/Lex [1] and yacc/jay [2] have always greatly simplified the job of a compiler writer. Unfortunately, if they are targeted to Java they usually require installation and the use of a build tool such as make [3] or ant [4]. Moreover, generators only produce Java source code which still needs further compilation, creation of a runtime environment, etc. The Java Software Development Kit [5] is a 57 mb download and expands into about 1700 files consuming 100 mb disk space. Does the proverbial Hello World example justify all this?

Wcs currently accepts source texts for small interpreters and Java programs and provides server-side execution of several compiler generation tools and Java compilation. The resulting class files are merged with runtime support, archived, and sent to the client as an applet where the interpreter or other program can be executed, see figure 1. The class System is spoofed in such a fashion that typical programs need not be adapted for execution in the applet.

Wcs is implemented as an HTTP servlet which processes GET and POST requests. The request parameters are designed to fit HTML form elements, i.e., a web page with one or more textarea elements can be used as a trivial editing environment for programs, see figure 2.

Of course, requests need not involve a textarea or even a form. A request can be embedded as a hyperlink or button within tutorial text and provide compilation and execution of a canned example. A more interesting tutorial is structured around an initialized textarea element and suggests changes that the user might want to make to the source text before it is submitted for compilation and execution.

Wcs is not limited to compiler construction. Requests can ask for Java compilation only and permit quick experiments with Java programs while eliminating the need to install the Java SDK on the client machine. We have created a number of Java tutorials for our introductory programming sequence and are currently testing user acceptance of this mode of instruction.

Requests
A service is requested through a set of key/value pairs which can be specified in any order. Compilation or preprocessing is described by keys to select a processor, provide source text, select a format for the result, and set processing options. These keys are grouped by a numerical suffix so that several compilation and preprocessing steps can be combined into one executable program. All keys are described at [6].
For example, a request with the following pairs returns an applet page to execute two public Java classes:

```
processor.0=javac
source.0=http://doc-files/A.java
sink.0=applet
processor.1=javac
source.1=doc-files/B.java
format.1=url
sink.1=applet
howto=doc-files/3.xml
```

processor and sink describe the input language and the desired output format — a parser generator such as javac can be asked to output a symbolic description or uncompiled Java code. The value of source is either the actual source text or, specifically for tutorials, a URL indicating where the text can be found; format might be needed to force the interpretation as an address.

In this example with two classes there could be two main methods. The applet page contains a drop-down list with suitable classes; more could be added with the key main. In a tutorial a value for howto is specified to insert execution instructions into the applet page.

Usually, the key/value pairs are sent by submitting a form to the web server. For convenience, wcs also supports the creation of typical form pages, optionally with Javascript-based dynamic creation and resizing of multiple text areas as shown in figure 2. form=true requests a form, the other key/value pairs set the default content of the form. A value for dynamic can limit for which processors textarea elements may be created dynamically. Forms created by wcs constitute a very simple IDE and can be a starting point to integrate requests with tutorial materials.

**Compilation**

The wcs servlet is executed in a Java-based container such as tomcat [7]. Java-based processors such as the Java compiler or a generator like JLex [1] could be executed directly within the servlet. However, to simplify development and accommodate native processors such as javac [2], a uniform approach was implemented based on the Runtime and Process classes of Java. Classes from a small Task framework encapsulate processor execution as a separate host process and implement input/output handling as strings and in temporary files.

Java sources are scanned with a regular expression designed to extract package and class name information and are inserted into appropriate directories in a temporary source tree. Java compilation follows all preprocessor executions, if any, and results in a temporary class file tree which is archived in a jar file.

The applet page is composed with an XSLT script which can easily include information from a request-specific XML file supplied with howto. Once the applet page is sent to the browser, the browser will ask the web server for the applet’s archive. At this point the servlet returns the jar file and deletes it from the server. While this approach makes it impossible to reload the applet page, it eliminates the need for garbage collection among files on the server.

**Execution**

Figure 2 illustrates that a command line can be specified at the top of the applet. The start button doubles as a stop button when the applet is running and should be able to curtail a runaway situation. In extreme cases the browser page may have to be closed.

The applet contains a text area which is connected with a pseudo-terminal simulation to the program in place of System.in, .out, and .err. This would normally require signing the applet and a policy file at the client. However, a programmer is not likely to fully qualify a reference to System. Therefore, the compilation process inserts a System class into every package it finds and all these classes delegate to the applet implementation. This is usually sufficient to support typical console input and output without a need for client-side preparations.

Unfortunately, paste operations do not work very well, neither with the AWT nor with the Swing classes. The AWT seems to insert pasted text into the graphic interface but not into the buffer visible to the program. Swing works a bit better but requires a specific permission. Therefore, the applet is based on Swing and supports pasting as long as the permission is granted.

Program execution could pose a security problem, but applet execution always happens on the client and in a sandbox that normally prevents any access to the local platform and network connections only to the host of the applet. Preprocessing and compilation options are limited so that a client should not be able to cause damage to the server (other than some denial of service if huge compilations are requested). Conversely, even if wcs sent malicious code within the applet the sandbox should be able to protect the client.

Some things cannot be faithfully simulated in an applet: the sandbox limits which kind of networking experiments are possible and without additional permissions there can be no file operations. Swing can be used, but windows may stay behind until the browser window is closed — there appears to be no reliable way to terminate platform-specific background threads such as those involved in Swing operations.

A more serious drawback concerns static initialization: this happens in a Java program immediately prior to the call to main whenever the program is executed. Currently the applet does not use a class loader and only loads classes once and some before the start button is pressed. As a consequence, static items are not reinitialized for subsequent start operations and output from static operations is not captured into the text area. One should argue that programs relying on this behavior tend to be in bad style.
Applications

The proof of concept implementation for wcs was used to implement a fairly extensive tutorial on language processing [8] which has been used successfully in courses on programming language concepts and compiler construction in the previous academic year (2004/2005). Wcs acts as a back-end for a tutorial on it’s own use [6].

More interestingly, wcs is currently used as a learning aid in an introductory programming course via a set of interactive web-based tutorials. The tutorials are intended to help students who are new to programming learn basic Java syntax, program structure, programming methodology, and problem-solving techniques. The tutorial topics currently include primitive vs. reference data types, conditional and repetition statements, class structure and method invocation, modifiers, arrays, and the software development life-cycle. Advanced topics such as inheritance, interfaces, the Java Collections Framework, inner classes, exceptions, and event-driven programming may be added in future releases.

We are conducting a study to gauge the effectiveness of the tutorials as a learning aid. A total of 226 first-year students are participating in the study, covering seven sections of Computer Science 1. To gauge effectiveness, two versions of the tutorials were developed – one using wcs to supplement the tutorial text with an interactive demonstration of the topic [9], and another version without wcs that provides the same text and a sample program, but no interaction [10].

At the start of the study, students were randomly assigned to one of the versions, with an approximately equal number of students assigned to each version. Students are instructed to visit the website for their assigned version each week, when a new tutorial topic is posted. A total of nine tutorials are planned for the completion of the study. The hypothesis for the study is that students who learn techniques and concepts with the aid of the interactive tutorial will have a better understanding of those concepts and will display a higher level of skill when applying those concepts than those who learn from a textual explanation combined with a static example.

Quantitative data is collected in the form of weekly quizzes that cover key concepts discussed in the tutorials. Each quiz consists of two questions – the first question involves either writing a Java code segment or predicting the output of a given code segment, and the second question is conceptual in nature, i.e., it is designed to test basic understanding of a key concept presented in the textual portion of the tutorial. The purpose of the two questions is to produce separate measures of the students’ ability to comprehend key concepts and their ability to apply knowledge of Java syntax and program structure. It is possible that the students’ use (or non-use) of the interactive examples will affect those measures in different ways.

As an additional dimension to the study, the effectiveness of the interactive tutorials is measured across different categories of students. For example, four of the seven sections of Computer Science 1 consist of a traditional lecture/lab format of instruction (152 students) whereas the remaining three sections consist of a studio format of instruction (74 students). The studio format is inherently more interactive than the lecture/lab format in that students receive all instruction while they are seated at a terminal, logged into their account, and presumably writing and testing code as it is presented in class. It would be useful to know if interactive tutorials based on wcs are more or less effective for students who are already acclimated to interaction.

Summary

The web compiler service supports parser generator tools and Java compilation on a web server and execution of the resulting program from an applet in a web page. It is useful for quick experiments and presentations without the need for installing a software development environment on the client side. wcs can be used to support interactive, web-based tutorials, e.g., on language processing and Java programming. We are using it in conjunction with a set of web-based tutorials as a learning aid for students who are new to Java. The effectiveness of the approach is being determined via quantitative data from quizzes taken by students who either have or have not had access to the interactive part of the tutorials. The results from the study will be presented in a future paper.

The development of wcs was supported in the summer of 2005 by a Provost’s Learning Innovation Grant from Rochester Institute of Technology.

References