Tcl and Java Integration

Ray Johnson

Sun Microsystems Laboratories
901 San Antonio, MS UMTV-29-232
Palo Alto, CA 94303-4900
rjohnson@eng.sun.com

February 3, 1998

Abstract

This paper describes the motivations and strategies behind our group's efforts to integrate the Tcl and Java programming languages. From the Java perspective, we wish to create a powerful scripting solution for Java applications and operating environments. From the Tcl perspective, we want to allow for cross-platform Tcl extensions and leverage the useful features and user community Java has to offer. We are specifically focusing on Java tasks like Java Bean manipulation, where a scripting solution is preferable to using straight Java code. Our goal is to create a synergy between Tcl and Java, similar to that of Visual Basic and Visual C++ on the Microsoft desktop, which makes both languages more powerful together than they are individually.

1 Introduction

In the simplest of terms, the goal of our project is to make Tcl the scripting language for the Java™ platform. We envision a dual language architecture where Tcl and Java work together in a seamless way. Tcl and Java are very well suited for this integration. In fact, both languages share many of the same design goals such as platform independence and the ability to run untrusted code. This white paper points out many of the reasons why Tcl and Java integration will make both languages more powerful.

Despite the languages shared philosophies, it is important to note that these two languages have historically had very different uses. The Tcl[1] scripting language is a classic scripting language, ideal for embedding into other applications. Tcl started out in the UNIX® world as a way to easily create tools with a command line interface. Tcl was first introduced in 1990 by its creator, John Ousterhout, a professor at U.C. Berkeley. In 1994, the Tcl project moved from Berkeley to Sun Microsystems Laboratories where ports to the Macintosh and Windows platforms were started. The goal was to make Tcl a cross-platform scripting language. Today, with some 500,000 users, Tcl is one of the most popular cross-platform scripting languages.

Java[2], on the other hand, is a new twist on the classic 3rd generation programming language (3GL). While the Tcl project was just getting underway, Java, created at Sun Microsystems by James Gosling, started to make its incredible introduction. Java soon became the biggest entry of a new programming language the industry has ever seen. It wasn't long before people were questioning the role of Tcl with respect to Java. This question was difficult to answer at first (we had to learn Java to really answer the question). However, it was obvious that the languages were nothing alike. Tcl was geared for small scripts, rapid application development (RAD), and dynamic environments. Java was geared for programming larger more complex applications. It is only now that many IS developers and other customers of Java are seeing the need for a scripting language within the Java platform. We aim to provide that.

The goal of our project, however, is much more than just providing another language alternative for the Java platform. Rather, we aim to create a tight synergy between Tcl and Java. The idea is to create an architecture where the developer can use the right tool for the right job. The architecture is a dual-language environment that is meant to support two very different communities of developers. Java should be used for creating components, frameworks, and other core technology that requires writing large detailed systems. Tcl should be used as glue code, a control language, or other smaller tasks that require a good deal of dynamism or high-level control. But most importantly the developer can communicate seamlessly between the two languages. The result is that together Java and Tcl are more powerful than they are individually.

Our current deliverable technologies include four items. Jacl is a 100% Java implementation of Tcl. Tcl Blend allows the (C based) Tcl interpreter to load and interact with the Java VM and vice versa. The Java Package, which is part of both Jacl and Tcl Blend, provides the communication between Tcl and Java. Finally we are working on a Tcl Bean for Java™ Studio™ that adds the power of Tcl for connecting Studio beans. We discuss each of these projects in greater detail later in this paper. We also suggest that you check out our home page[3] for the most up-to-date information about our projects.

The rest of this paper is outlined in the following way: Section 2 discusses the various motivations behind our project from different perspectives. Section 3 focuses on the Java Package and the synergy we attain by integrating Java and Tcl. Section 4 discusses some possible applications of this technology. Our implementation strategy and current plans are outlined in Section 5. Finally, Section 6 discusses related work and Section 7 wraps things up.

2 Motivations

This section discusses the various motivations for integrating the Tcl and Java programming languages. Integrating Tcl and Java brings new power to both Tcl and Java developers. While describing our motivations, this section explicitly describes the motivations from the unique points of view of both Java and Tcl programmers.

2.1 What Tcl provides for Java users.

A new language for building Java applications. Java is best classified as a system programming language. It is ideal for building components from scratch and has all the complex data structures, type safety, and other features a developer needs to build large systems. The complexity in components is in the representation and handling of data structures and algorithms. For many tasks, however, a scripting language like Tcl is a much better tool. Scripting languages are designed for building small to medium sized applications that focus on rapid development or dynamic behavior. Here, the complexity is in the connections. Scripting languages are ideal for "gluing" together existing components or applications. For a more detailed discussion on the trade-offs between systems programming and scripting, please see a white paper written by John Ousterhout on the subject. [4][5]

Add scripting to Java applications. Tcl has an interesting attribute not found in many scripting languages: It is a library that is easy to embed into existing Java applications. By using the Tcl library, application developers can add scripting capabilities to their Java based applications. It is easy to extend the Tcl language to provide very high level, application specific commands that interface a product's specific application features to the Tcl language. By adding scripting capabilities to their applications, Java developers can add a great deal of power, flexibility and end user customization to their Java applications. In addition, having scripting capabilities available across a suite of Java applications provides a way to tie those applications together, much like Visual Basic for Applications (VBA) does for Microsoft Office. Figure 1 shows how end users could even create new custom applications by using Tcl to access the functionality of scriptable Java applications. Using the Tcl library, Java developers can add scripting capabilities with the least amount of effort.

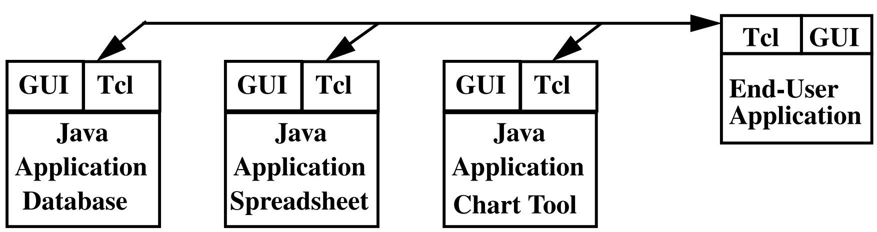

Figure 1. Tcl can be used to give a suite of Java applications a common interface for extending the applications and for end-user customization. This notion is similar to how Microsoft extends it's Office application suite with VBA.

A better interface to JavaBeans™ components. The benefits described above are the same as the benefits a C programmer enjoys by using Tcl. We are, however, working on several features that are Java specific and, in fact, are only possible because of the specific features of the Java environment. We have created new interfaces to script the powerful JavaBeans component mechanism. Tcl has always been a great language for gluing systems together. We expect that defining the connections between components will be 5-10 times faster with our Tcl interfaces than writing the connections in Java. There are a variety of reasons for this that the paper discusses in fuller detail in Section 3. Any development organization interested in doing work with JavaBeans will certainly want to evaluate Tcl as a tool for gluing them together.

A tool for prototyping and testing. In addition to our Beans interfaces, we have also developed a general interface to the Java Reflection API's that makes Tcl ideal for prototyping and testing Java applications. Our interface allows Tcl code to dynamically create and destroy objects in the Java VM. Tcl can also set the fields for and call the methods on any Java object. This gives incredible power for interfacing Tcl code to the rest of the Java environment. Among other things, a Java developer can use this feature to rapidly prototype a Java module in Tcl, change features quickly and on the fly, and once they have decided upon an actual feature set, rewrite the code in Java for maximum performance. The reflection interfaces are also ideal for testing Java code. It's easy to write Tcl code that can call every method in your API and test error conditions and return results. We give an example of using Tcl for testing in Section 4.2. For now, let's just say that depending on the kind of work Java developers need to do, Tcl can be a vital tool for accelerating the development process.

A bridge for porting legacy code. Tcl has always been good at gluing together various systems into an integrated whole. Tcl features and extensions like exec, expect[12], and TkSteal, are ideal for wrapping legacy systems into new systems. Tcl Blend allows this same power for integrating C based code into the Java platform. With Tcl Blend developers use the Tcl scripting language to glue together legacy code with new Java systems. The illustration in Figure 2 shows how Tcl Blend gives developers a clear path for porting C code to become 100% Java in an incremental way. Over time, the developer can incrementally port their legacy applications to Java. When all of your C based code is finally moved to Java, the developer can switch from the C based Tcl to using Jacl. The end product is a completely Java based solution.

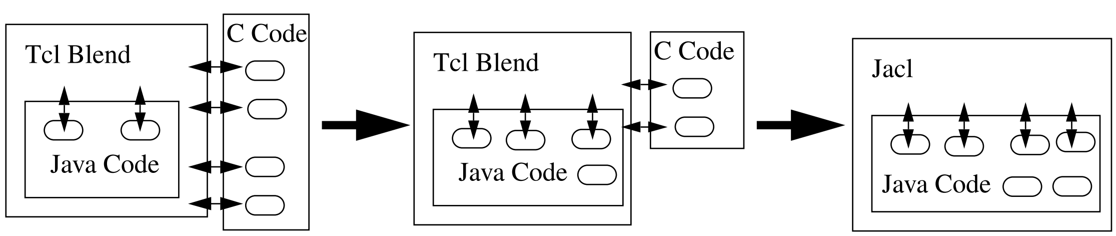

Figure 2. This figure shows the incremental transition to Java by using Tcl Blend to integrate new Java code and legacy code to eventually using Jacl for a 100% Java solution.

2.2 What Java provides for Tcl users.

Cross-platform extensions. Tcl was originally designed as a library for quickly creating a new scripting language for a given tool. As such Tcl has always been a two language system. The basic Tcl library contains a standard interpreter, control structures, etc. In addition, the language was designed to be extended with interesting functionality via C or C++. As the language has grown it has shifted to being a cross-platform scripting language. Unfortunately, the extensions that people wrote were written in C and, as a result, were not portable. Java provides us with a way to extend the Tcl interpreter in a language that is cross-platform. By using Java, users can write extensions that are cross-platform, a feature long desired by Tcl users.

Tcl runs on more platforms. The portability of Tcl has always been a strong draw to the language. Tcl already runs on Macintosh 68k & PowerPC, Windows 95 & Windows NT, and all flavors of UNIX. However, by implementing Tcl in the Java programming language, we allow Tcl users to run Tcl code on the Java Station and in web browsers. Furthermore, Tcl will run on any new platform that Java is ported, sparing the Tcl developer the trouble of porting. This is of particular value considering the amount of work going on with respect to Java environments.

Tcl can take advantage of Java features. Tcl will be able to take advantage of many of the numerous advanced API's being added to the Java platform. For example, Tcl could use the new 3D API's to add new widgets to the Tcl/Tk toolkit. The benefit is that these API's would be cross-platform and the extension writer would not have to do a new implementation for each platform. Furthermore, the Java Package will make using the Java API's possible directly from Tcl without writing new Tcl commands.

2.3 Tcl and Java Synergy: A new architecture

Aside from the specific benefits Tcl may provide to Java users or the benefits Java may provide to Tcl users, there are additional benefits that come from the languages working together. These benefits are made more profound if you notice a split that is occurring between different communities in the developer world. The split separates traditional, 3GL based, "component developers" from higher level, scripting centric, "application assemblers".

What we call "component developers" generally include C, C++, Java and other 3rd generation language (3GL) developers. These developers tend to develop code from scratch or generic frameworks. They primarily build operating systems, large applications or large reusable components. The tools they use (including the languages) focus on performance, memory management, protocols and other low-level system characteristics. The important engineering choices of these systems center around data structures and algorithms.

"Application assemblers", on the other hand, consist of developers that use scripting languages, visual programming applications, and other 4GL tools. Rather than coding from scratch, these developers tend to "glue" together large modules or applications (often written in a 3GL). That is, they might quickly assemble applications out of large pre-existing components, or perhaps they will wrap existing legacy systems and get them to work with new systems. Low-level system characteristics are usually not a concern with these types of applications because the amount of code is usually really small and often dwarfed by the larger components. Rather, application assemblers look for tools that have flexibility of communication, can deal with types dynamically, or have other features that aid in the gluing of a myriad of different components and applications. Here, the important engineering choices focus on the connections between components. While the systems assembled may be quite large, the amount of code written tends to be much smaller than what's written by systems developers.

For the Java platform, we envision an architecture that includes Java as the "component" language used by component developers and Tcl as the "glue" language used by application assemblers. We believe this is an ideal match. Java has all the characteristics wanted in a systems language. In addition, with features like JavaBeans, the memory model and the Reflection API, Java is a better language than C or C++ for building reusable components that can be manipulated by other tools. On the flip side, Tcl is an ideal "glue" language because of its dynamic types, flexible runtime model, and extensibility. We have created the Java Package to make this match even better.

This dual language approach to development is not new. Microsoft has used the approach very effectively on the Windows desktop. Windows developers use Visual C/C++ for systems programming and the development of components. They use Visual Basic (VB) for scripting components into small to medium sized applications. VB users buy (reuse) components from developers that specialize in developing components. The VB users quickly write code that combines these components into small to medium sized applications. This model has worked very well for Microsoft and Microsoft's customers.

The problem of course, is that Microsoft has used Visual C/C++ and VB to lock people into the Windows desktop environment. Java and Tcl, on the other hand, have focused on being cross-platform which is essential for network based computing. In all other ways, however, the relationship is similar. VB can be used to script ActiveX components. Tcl can be used to script JavaBeans. C based applications can be made scriptable by Visual Basic for Applications (VBA). Java based applications can be made scriptable with the Tcl library. In short: Java and Tcl are to network computing what Visual C/C++ and Visual Basic are to the Microsoft desktop.

3 Scripting Java: The Java Package

This section focuses on the specific ways in which Tcl has been extended to script the Java VM. These are specific features that stand Tcl and Java integration apart from just another port of a programming language to the Java platform. We will not, however, discuss the specific benefits of Tcl or Java per se which have been dealt with better in other contexts [1][2].

The sum of these features that integrate Tcl with Java is something called the Java Package. The Java Package is available in both Tcl Blend and Jacl. They are a set of Tcl extensions that interact heavily with the Java VM. The Java Package works with JDK™ 1.1 or later.

3.1 Using the Java Reflection API's from Tcl

As of JDK 1.1, Java has had a new reflection API which makes more dynamic use of the Java VM a possibility. The reflection API's also make it possible to script Java. Scripting

Java means that the developer can dynamically determine methods and fields of an object or class. They can call or manipulate those methods and fields. But most importantly, they can do all those things during runtime without having prior knowledge of which objects and classes will exist during runtime.

The following is a list of the types of things we can do with the Java Package thanks to the Reflection API's in JDK 1.1. All of these features will also be in the 1.0 release of Jacl and Tcl Blend.

- Create new instances of any public class including arrays.
- Call methods on objects or static methods for a given class.
- Access or set any public field of an object (or class if field is static).
- Determine the class, base class or super class of an object.
- Determine the public methods and fields available for a given object or class.
- Load new Java classes from a class path specified in Tcl.
- Define new Java classes from a byte stream in Tcl.

While the list is not complete, it gives a good idea of the power of the Java Package. Furthermore, using this power from Tcl is vastly easier to do than using the reflection API's directly. In short, one can do from Tcl, via the Java Package, what one can do in Java. The example in Figure 3 gives a very simple example of the Java Package in action.

```
set f [java::new java.awt.Frame]
$f setTitle "Hello World"

set text [java::new java.awt.TextArea]
$text setText "Simple example of Tcl scripting Java."
$f {add java.awt.Component} $text

$f setLocation 100 100
$f pack
$f toFront
```

Figure 3. This figure shows the Java Package creating a simple user interface using the AWT toolkit.

3.2 Scripting Java Beans

In addition to the Reflection support, the Java Package also supports the manipulation of JavaBeans. JavaBeans are the component mechanism used in the Java platform. However, in its simplest form Beans are just classes that follow a set of naming conventions.

The Java Package includes several ways to manipulate JavaBeans. Here is a short list of the kinds of things that are possible with the Java Package:

- Create new beans and call any method on the bean.

- Set or get any property defined for the JavaBean.
- Determine the list of events supported by the bean.
- Register Tcl call backs to bean event listeners.

The last feature is perhaps one of the most interesting. Java developers must ensure that they have the right type of object interface before they can register interest in an event on a particular bean. This usually involves writing new “adaptor” classes that glue the two objects together. With Tcl’s Java Package however, the developer simply needs to use the java::bind command to register a Tcl script with a new Java object. Only one line of code! (See Figure 4 for a simple example.) Of course, we are doing the extra work of creating the adaptor classes behind the scenes. The result, however, is that beans are easier to use.

```
set f [java::new java.awt.Frame]
set b [java::new java.awt.Button "   Exit   "]

java::bind $b actionPerformed {set done yes}

$f {add java.awt.Component} $but
$f setLocation 100 100
$f pack
$f toFront

set done "no"
vwait done
$f dispose
```

Figure 4. This figure shows the use of the java::bind command to register a callback with the Button bean. Clicking the button will set the variable “done” which will allow the script to continue and dispose the Frame.

3.3 Future enhancements to the Java Package

We are still researching ways in which we can improve the Java Package to make scripting Java from Tcl even easier. Some of the features we are currently planning include implementing Java interfaces in Tcl and subclassing a Java class with Tcl procedures. We are also considering generating complete JavaBeans that are implemented in Tcl. Java applications using these Tcl-generated beans would not even know they are using beans that were developed with the Tcl scripting language.

4 Applications

The number of possible applications of Jacl and Tcl Blend are limitless. This section gives a few important examples that we feel may be early applications of our technology. Many of these application areas are traditional areas of expertise for the Tcl language. However, some will leverage this new synergy we are creating between Tcl and Java.

4.1 Web Server CGI

One of the largest growth areas for Tcl is with CGI scripting. Many people are finding Tcl easier to learn, use, and maintain than Perl. Now that Jacl brings the power of Tcl to the Java platform, we expect Jacl to suit the CGI needs of Java based web servers.

4.2 Testing

Tcl is often used for writing white box test scripts. It is ideal for testing little snippets of code for an expected outcome. With our use of the Java reflection API's Jacl is even better suited for this type of application[6]. The current Tcl implementation has about 211,000 lines of test code written in Tcl. The existing Tcl test suite has enabled us to port Tcl to Java very quickly and has also assured us that our implementation is rock solid. Other developers will find our testing framework easy to adapt to their own testing needs.

Figure 5 contains an example pulled directly from our test suite. Java code can be called and manipulated directly with our Java scripting interface. The test command is part of the testing framework which we ship with our source distributions for Tcl.

```
test invoke-11.8 {getClassByName} {
    set k [java::new {java.lang.Integer int} 1]
    java::instanceof $k java.lang.Integer
} 1
```

Figure 5. Each test runs a script and compares the result to an expected result. This test is making sure a created object is of the proper type.

4.3 Small IR Applications

Tcl has always been ideal for gluing together functionality to quickly create useful applications for the enterprise. Much of this is due to Tk, which is used to add a GUI to legacy applications or complex systems. Unfortunately, Tk has yet to be ported to Java. However, with our support for Java Beans, Jacl is already a very useful tool for constructing prototypes or full featured applications.

4.4 Scripting Applications

Tcl was invented to be embedded in applications to provide them with a scripting interface. Jacl is a library that can be used by Java application developers to add scripting features to their applications. Jacl is far and away the easiest and most flexible way to add scripting to your Java based applications. (See Figure 1 on page 3.)

4.5 Transitioning to the Java Platform

Our native Tcl implementation of Tcl Blend allows developers to migrate to Java without losing functionality available in legacy (or C-based) applications. Tcl Blend allows the developer to create "hybrid" systems implemented in both Java and C in a much easier manner than using the JNI (Java Native Interface) directly from within C. Over time, C based code can migrate to Java. Eventually, Tcl Blend can be replaced with Jacl and the result is a 100% Java solution. Developers looking for a way to integrate Java with their legacy systems should really consider the power of Tcl Blend. (See Figure 2 on page 4.)

4.6 Writing new components: Java Studio

We are currently working on a Java Studio bean component that is based on Jacl. With the Tcl component, Java Studio developers can write new behavior for Java Studio directly with Tcl. We are planing to eventually allow developers to write new Java Beans completely in Tcl. Java users can use those Tcl beans directly without ever even knowing that the beans are implemented with Tcl. Figure 6 contains an example of a Java Studio bean written in Tcl that mimics the Memory bean that ships with Java Studio. Note how little Tcl code is needed!

```
# Set the following list to hold the memory locations for this bean.
set memory_locations {store1 store2}

foreach x $memory_locations {
    studio::port in "$x input" -transfer dynamic
    studio::port out "$x output" -location east -transfer dynamic
}

studio::port in Trigger -location north
studio::bind Trigger {
    foreach x $memory_locations {
        upvar #0 "$x input" input
        upvar #0 "$x output" output
        if {[info exists input]} {
            set output [$input toString]
        }
    }
}
```

Figure 6. This example Studio Bean creates input and output pins for each memory location. A trigger pin is also created that, when fired, will transfer any data that came in the input pins to the output pins.

5 Implementation

This section will discuss our implementation strategy and give a rough roadmap for construction. The products we are currently constructing are Tcl Blend, Jacl, and the Tcl bean for Java Studio. This section will discuss our implementation plans for each project.

5.1 Java & Native Tcl: Tcl Blend

One reason for doing Tcl Blend is to get the full power of Tcl and Java working together in the shortest period of time. Tcl Blend allows the user to use all of Tcl and Tk and all of the features of Java. Tcl Blend will be feature complete long before Jacl. This is good news for Java developers as well. They will be able to test their Tcl Java code now and know that there is a planned roadmap to make that 100% Java in the future.

Tcl Blend is progressing smoothly. The biggest task will be to get Tcl Blend to work everywhere Tcl and Java 1.1 both exist. This will be an on going job that will be customer driven. We also still need to integrate Tk and Java window systems to allow the two to work together. Even with the limited resources we currently have available for Tcl Blend, we should be feature complete within calendar year 1998.

5.2 100% Java: Jacl

Of course, for many users it is important to have a 100% Java solution. Our long term goal is to port all of Tcl & Tk to the Java programming language. However, it must be remembered that Tcl and Tk are very large systems and a full port will take a great deal of effort. Tcl is implemented in some 80,000 lines of C code. Tk is another 120,000 lines of C code. Our current estimates for a full port is around 6 to 10 man years. We are currently staffed with three full time engineers and are looking at an elapsed time of around three years. The work is such that we can do a lot of things in parallel and we could dramatically reduce the schedule if given additional man power.

The good news is that the core of Tcl has already been ported and is extremely useful. The core includes all of the control structures for the language, basic file support, and other important features like regular expressions. Because Tcl has such a large test suite we are also assured that the code ported so far is very stable and completely compatible with the C version of Tcl. We plan to release version 1.0 of Jacl in the first quarter of 1998. While this will represent only a subset of Tcl, it will be of commercial quality.

After this initial release, we will be making regular releases that will incrementally increase the subset of Tcl and Tk that Jacl supports. Exactly what those features are will depend heavily on customer demand. We expect that we will have bi-annual releases until Tcl and Tk are completely ported.

5.3 Strategy for porting Tk

Due to the length of time it will take to port all of Tk we are looking for ways to reduce our schedule yet still retain the most value. Specifically, we want to find a way to get the value of Tk without having to do a full port. It turns out that a majority of the code in Tk is in the widgets themselves. Just the Text and canvas widgets alone make up roughly one third of all of Tk. We also would like a way to better integrate the Tk widget set with the AWT widget set.

Instead of doing a straight port of Tk we plan to “wrap” existing AWT widgets (beans) in such a way that they look and behave like Tk widgets. In fact, we hope to be able to create a mechanism where any Java bean with a GUI can be used by Tk directly. Tk will get the benefit of a large set of reusable widgets without our having to write or port too much code.

However, many of the widgets in Tk - especially the canvas and text widgets are vastly superior to the AWT widgets. We will eventually need to port these widgets in order to obtain the full benefits of Tk. We hope to port these Tk widgets as Java beans so that they will be useful outside of the context of Tk as well. For complete compatibility we will also need to create new Beans for the basic widgets like Buttons and Scrollbars. The reason is that the Tk widgets (for the most part) have a much richer feature set.

5.4 Tcl Bean for Java Studio

The first version of the Tcl Bean for Java Studio is nearing completion. In fact, we hope to release the bean with Jacl 1.0. We plan to evaluate customer response to the Tcl Bean before planning our future feature set. Our goal is to get the Tcl Bean bundled with Java Studio. Contact me directly if you have questions about this product.

6 Related Work

Tcl is not the only language being ported to the Java VM. Some of the other projects we know of are included below. We also discuss the differences between the work they are doing and what we are trying to do. Please remember that this list is only a snapshot in time and information about various efforts is sketchy at best.

Perl. Perl[8] is one of the more well known scripting languages. Traditionally Perl has been used for data conversion or processing but has recently dominated the world of CGI scripting. It is rumored that O'Reilly & Company's "Perl Resource Kit" has some limited support for Java.

Python. Python[9] is also a cross-platform scripting language that has been around for a few years. While it hasn't attained the wide spread use of Tcl or Perl, it certainly has an avid following. Currently there is an effort to port the Python scripting language to Java called JPython[10]. In fact, their approach is to compile Python code directly down to Java byte codes. The JPython work currently doesn't include anything like the Java Package but their work, like ours, is on going.[13]

JavaScript. One of the most common questions we receive is, "how does this relate to the JavaScript[11] programming language?" The simple answer is that JavaScript was created for, and has always focused on web page scripting. JavaScript is not geared for, nor has been used to any extent for application development. Tcl, on the other hand, is focused on creating or working with applications. As far as we can tell, Netscape has no clear plans to make JavaScript a general purpose scripting language.

Visual Basic. Visual Basic has some efforts to make Visual Basic work with Java and JavaBeans. However, their focus remains on supporting ActiveX. Current offerings of Visual Basic are also not implemented in Java and are not cross-platform. Nonetheless, with a company as large as Microsoft, you must closely watch what they do in this space.

7 Conclusions

This paper clearly points out motivations, feasibility, and benefits of integrating the Tcl and Java programming languages. Depending on the customer, a scripting solution for the Java platform may range from very useful to outright essential. Our initial efforts have been received with high praise, and we have a clear path to providing a complete solution. Some customers are already receiving the benefits of this work, and more are trying it every day. Any serious developer working on applications for the Java platform will certainly find the synergy between Tcl and Java a great tool for addressing their development needs.

8 References

[1] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, ISBN 0-201-63337-X, 1994.

[2] K.Arnold and J. Gosling, *The Java Programming Language*, Addison-Wesley, ISBN 0-201-63455-4, 1996.

[3] Jacl & Tcl Blend home page: http://sunscript.sun.com/java/

[4] J. Ousterhout, *Scripting: Higher Level Programming for the 21st Century*, `http://www.sunlabs.com/people/john.ousterhout/scripting.html`.

[5] J. Ousterhout, *Additional Information for Scripting White Paper*, `http://www.sunlabs.com/people/john.ousterhout/scriptextra.html`.

[6] S. Stanton, "TclBlend: Blending Tcl and Java", *Dr. Dobb's Journal*, pp. 50-54, Feb 1998.

[7] B. Kernighan and D. Ritchie, *The C Programming Language*, Second Edition, Prentice Hall, ISBN 0-13-110362-8, 1988.

[8] L. Wall, T. Christiansen, and R. Schwartz, *Programming Perl*, Second Edition, O'Reilly and Associates, ISBN 1-56592-149-6, 1996.

[9] Watters, A., G. van Rossum, and J. C. Ahlstrom, *Internet Programming with Python*, MIS Press/ Henry Holt Publishers, 1996.

[10] J. Hugunin, *Python and Java: The Best of Both Worlds*, <URL:http://www.python.org/jpython/ipc6paper.html>

[11] D. Flanagan, *JavaScript: The Definitive Guide*, Second Edition, O'Reilly and Associates, ISBN 1-56592-234-4, 1997.

[12] D. Libes, *Exploring Expect*, O'Reilly and Associates, ISBN 1-56592-090-2, 1995.

[13] Home page for O'Reilly Perl product: http://perl.oreilly.com/