IPv6 Monitoring web services

Scalability, Reliability and Robustness

Analytical Server Project
Masters programme on System and Network Administration
University of Amsterdam, the Netherlands

`http://pisa.toscana.os3.nl:8080/~jvb/6net/welcome.html`

Jeroen van Beek
jvb@os3.nl

Gert Jan Verhoog
gjv@os3.nl

2nd February 2004

Abstract

During a course on Web Services[1] for the masters programme on System and Network Administration[2] several students developed web services and clients [4] to visualise the number of advertised IPv6 prefixes on the Internet over time, providing an indication of the growth of the next-generation network layer of the Internet, which is based on Internet Protocol version 6. The current implementation of these web services has several drawbacks. Firstly, it is unclear whether the current web services are able to handle a large number of clients. Secondly, the web services are dependent on a number of external programs, among which are certain unix tools and a specific database implementation, thus limiting possibilities for easy deployment of the web services on other servers. The third problem is that the current web services have crashed several times for unknown reasons. Finally, we would like to expand the functionality of the web services. In this document, we attempt to analyse the necessary steps to increase the scalability, flexibility and robustness of the web services and in what way we could expand their functionality.

[1] Project Web Services, fall 2003
[2] University of Amsterdam, the Netherlands, http://www.os3.nl/

Contents

1 Background

The Internet consists of a large number of interconnected networks. Packets of information flow from one network to another via devices known as *routers*. Routers act like air traffic controllers for the Internet: They know which networks a packet has to travel through in order to reach its destination. Also, a router decides the path a packet has to follow. Internet providers *advertise* the networks under their control to other providers and network administrators, informing them of the networks that should be reachable on the Internet. The providers negotiate their traffic exchange with others in so-called *peering agreements* [1].

The internet providers exchange this routing information using the *Border Gateway Protocol*, or BGP for short [2, 3]. Routers that use BGP maintain a list of IP[3] *prefixes*[4] that determines reachability between internet providers, or *Autonomous Systems*. Each BGP router that is connected to the Internet maintains a full list of network prefixes in its *BGP table*. By looking at one router's BGP table, you get an idea of the number of networks on the entire Internet.

Currently, the BGP table for the IP version 4, or IPv4, Internet, contains roughly 125.000 networks. The BGP table for the IPv6 network, which is the network layer for the next generation of the Internet, contains slightly less than 500 networks. This number is growing.

The IPv6 web services software visualises the growth of the IPv6 Internet by examining the IPv6 BGP table daily. The software acquires the BGP data from publicly accessible BGP route servers, so-called *looking glasses*. The web services provide functions to access this information. In addition to this, functions are available to retrieve provider names and locations for AS-numbers and network prefixes. This information is retrieved from several *whois* registries: 6Bone[5], Apnic[6], Arin[7], Lacnic[8] and Ripe[9]

1.1 Information flow

The IPv6 web services process information from various sources and deliver it to its clients. The flow of information is as follows:

1. the parser connects to a looking glass router and retrieves a BGP table;
2. the parser filters out data that is irrelevant for the web services software;

[3] Internet Protocol, the network layer of the Internet

[4] a network prefix, "2001:610:148::/48" for example, denotes a network that consists of all IP addresses starting with "2001:610:148:". The example prefix denotes the network containing IP address of the www.6net.org webserver, "2001:610:148:dead:210:18ff:fe02:e38".

[5] http://www.6bone.net

[6] http://www.apnic.net

[7] http://www.arin.net

[8] http://www.lacnic.net

[9] http://www.ripe.net

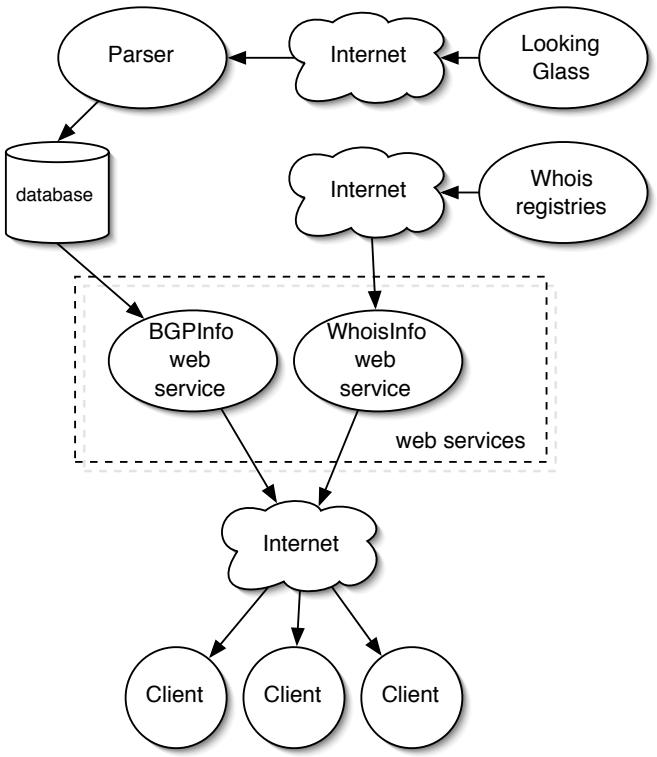

Figure 1: Information flow in the web services software

3. the parser stores the processed data in a database. These three steps are executed once a day.
4. The web services wait for incoming requests from clients;
5. the web services retrieve the requested information from the database and/or whois registries, process this information and return the results to a client.

The information flow from BGP router and wois registries to web services clients is illustrated in figure 1.

1.2 Elements

Based on the previous section, we identify the following elements that make up the web services software:

- One or more *Looking Glass* BGP router servers
- Parser software that stores the filtered BGP data in a database

- a database to hold the BGP information
- web service functions that retrieve and process stored information from the database
- web service functions that retrieve information from whois registries
- a server that provides access to the web services
- client applications that visualize the information they request from the web services.

2 Goals

The first implementation of the IPv6 web services [4] was developed for a course on Web Services for the masters programme on System and Network Administration by a group of students. The software was developed using Ruby [6] and SOAP4R [5] for the web services, Obective-C/Cocoa for the Mac OS X [7] client, MySQL for the database [9] and PHP for the web client and BGP parser [10].

We plan to improve this implementation in four areas: Availability, Scalability, Flexibility and data integrity.

2.1 Availability

Users of the web services should be able to access the functionality anytime, so the web services should be available all the time. The ruby implementation of the web services crashed a few times for unknown reasons. A source of the problems could be that the SOAP implementation for Ruby, SOAP4R, or the server, WebRICK, have not matured yet. We believe we could improve the availability of the web services by developing them using proven, mature technologies for programming language, SOAP server, and database systems.

2.2 Scalability

Scalability means how well a solution to some problem will work when the size of the problem increases. The IPv6 web services will be publicly available, for an indefinite time. This means that the software should be able to handle an increasing number of requests per time unit and a growing BGP table. We believe good scalability will only be possible if the web services are designed with this idea in mind. The core of the web services consists of a database system and SOAP functionality. The database must be able to handle an adequate number of transactions. The SOAP library wraps, transports and unwraps datastructures in XML format, which is fairly processor-intensive. Using a good SOAP implementation that handles these tasks well is very important.

2.3 Flexibility

It is not yet known how many users the web services will serve in the future, or how fast the number of advertised IPv6 prefixes in the BGP table increases. At this time we do not know what deployment options – in terms of hard- and software – will be available for the web services. We want the web services to be flexible: It should be easy to migrate to other environments. By developing the web services in a programming language for which a number of high-end SOAP servers exist, we ensure that deployment on one of these servers will be possible. Furthermore, by adding an abstraction layer between the web services and the database, using another database system in the future does not require large rewrites of the web service software.

2.4 Data Integrity

The first implementation of the IPv6 web services retrieve the BGP table from one looking glass router. Relying on a single source for this data presents a risk: Problems in the looking glass router, caused by network- or equipment downtime or configuration errors, will be copied by the web services. By using multiple – preferably not interdependent – sources of data, risks can be lowered.

3 Realization

3.1 Availability

To achieve high availability, we decided to use an application server environment with a proven track record. Unfortunately, the current web services implementation, which uses SOAP4R, a SOAP implementation for Ruby, has shown to be less stable than we would like. In addition to this, we would like to implement the BGP table parser in a more "professional" environment than PHP, which is what we are currently using.

Since IBM is one of the participants of the 6NET project, IBM's WebSphere [13] platform is a promising choice for the server environment. However, a WebSphere server was not available at the start of this project, so we had to make do with an alternative. To ease a possible future migration of the project to a WebSphere platform, we decided to implement the services in Java. Professional application servers for Java, such as JBoss and Tomcat are available [14, 15]. Having some experience in using the Tomcat server, we decided to choose Tomcat as our server platform during the development of the IPv6 web services.

3.2 Scalability

Good scalability can be achieved only by choosing the right tools for the job: We need a fast and scalable SOAP implementation and database system. There is a wide variety of database systems available, from MySQL, which is fast but limited

in its functionality, to high-end enterprise systems such as Oracle and DB2. At this moment, we don't know which database system the IPv6 web services will use in the future. By implementing the web services' database access using an abstraction layer known as a *database connector*, we ensure portability of our code to different database systems (see section 3.3).

The SOAP library is more or less determined by choosing Tomcat to deploy the web services. Tomcat uses the Java platform. The standard SOAP implementation for Java is the *jax-rpc* library.

3.3 Flexibility

We try to reach flexibility by limiting the number of external dependencies. We do this by using standard Java libraries only. In this way, the amount of platform-specific code is kept to a minimum.

To provide and manipulate data concerning the IPv6 network, data storage is needed. The decision is made to use an SQL-solution. Data is gathered by the parser application, will be formatted, and is submitted to a database. The 6NET environment might require large scale deployment. To make this possible and to prevent product or vendor lock in, the choice is made to make database access as transparent as possible. Application programmers do not need any knowledge about the used database systems to execute SQL queries. In case of under dimensioned hardware systems, easy migration to faster hardware is a must. To meet these requirements – transparent database access running Java – 'Java DataBase Connectivity' technology, JDBC, is used. The JDBC API provides cross platform connectivity for a wide range of data sources; from plain text file databases to enterprise level solutions. The issue of scalability of the database system will be moved to a level irrelevant to us; migration to a more advanced database system is one of the system features now.

3.4 Data Integrity

A significant part of the IPv6 web services use information directly or indirectly obtained from data stored in the database system. Integrity of this data is vital. It may occur that BGP routing tables contain invalid or incorrect information, e.g. by a configuration error. There errors may cause generation of unreliable information; an unwanted situation, which could be avoided by querying more than one server. Errors can be detected if two BGP routing servers are used although an error can be detected only. It is not known which routing table is right and which one is wrong. By using three routing servers, the chances of incorrect information are decreased significantly. If one server fails or provides the system with invalid or incorrect data, the error can be detected and the dataset can be ignored. The chance of using faulty information is reduced from P to P^2.

Observing the dataset we have collected so far from a single route server, we see the need for querying multiple servers; see the glitch in graph 2.

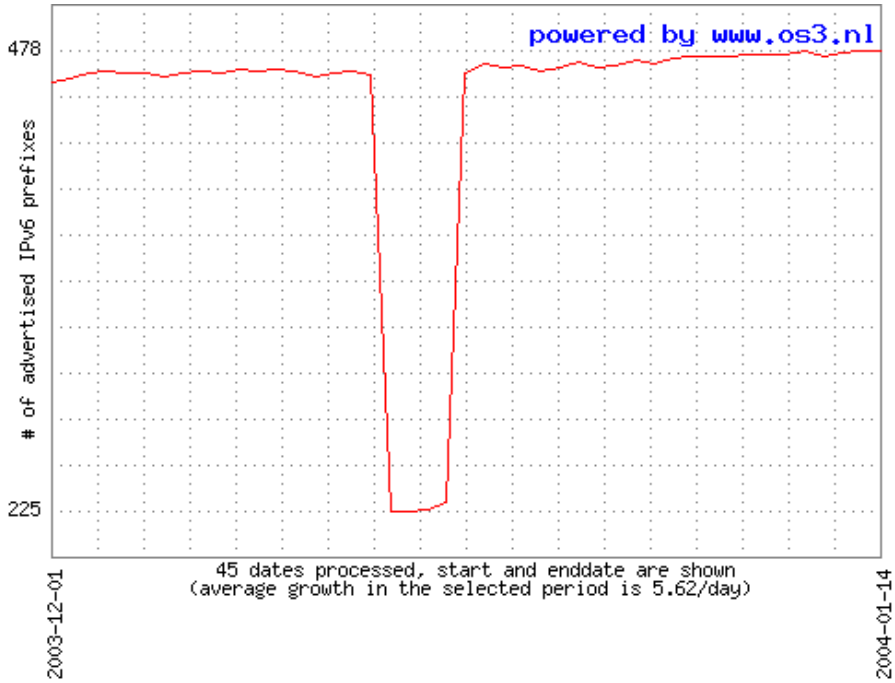

Figure 2: Number of IPv6 prefixes per day

4 Planning

The rollout of this project is divided in a number of phases which are described below:

4.1 Setting up a development environment

When the project started, there weren't any IBM WebSphere development platforms available to us. Because of the short lifespan of the project – four weeks in total – we decided to start working on an alternative development platform. Keeping the Java programming language in mind, we have chosen to develop the web services using Apache's Jakarta Tomcat and Sun's *Java Web Services Development Pack* [11].

The first step is installing Tomcat. After that the environment's *workflow* needs to be configured; a build-system which 'translates' Java source files into an enrolled web service. We use Apache's *ant*, a utility providing functionality like the well-known *make* application [12]. To start, a simple "HelloWorld" web service and client are coded and generated.

4.2 Researching needed technologies

For developing web services a number of technologies are required. Knowledge about these technologies is required to make adequate choices:

- Java technology for creating web services, especially *jax-rpc*.
- BGP router tables are imported using socket connections. Setting up socket connections using Java does not require external libraries.
- The current whois web service implementation uses an external console application ("whois"). The same functionality can be implemented using a socket connection.
- The graph showing the number of advertised IPv6 prefixes needs to be ported to a Java Applet. Needed functionality is available in Java's *awt* and *swing* libraries.
- The parser requires BGP tables as input. Output data will be stored in a database. The BGP web service retrieves this data *from* the database. As described in section 3.3, we do not want to communicate with the database directly to prevent a database specific implementation. Java's *JDBC* library accomplishes this.

4.3 Porting of existing functionality

After all required infrastructure and other components became available, Java implementation could start. Main target is implementation of *all* web services functionality in the Java language. Web service interfaces need to be compatible so existing clients still can be used connecting to the new web services.

4.4 "Must Have's"

After porting is completed, a number of other goals need to be accomplished. First of all, provided information needs to be reliable. To be able to collect reliable data, functionality must be added to the parser.

4.4.1 Integrity of whois information

The existing implementation parses whois information from different whois databases. In some cases, returned information is invalid. This is caused by a wide variety in database formats used in the whois database servers. The issue must be solved.

4.4.2 Integrity of BGP table information

As described in section 3.4 a method for detecting faulty BGP table information needs to be added to the system. This can be accomplished by reading out at least three different BGP route servers. Access to two extra looking glass servers is needed. New code must be developed to be able to compare routing tables and to detect possible errors.

4.4.3 More efficient database use

The existing implementation saves a snapshot of the BGP table in a database everyday. For some operations the existing web services need to process numerous database records, e.g. for showing added and deleted prefixes in a selected period. More efficient database design will increase performance and decrease response times of the system as a whole. Caching tables for the number of advertised prefixes need to be added. Scalability characteristics will be improved.

The old whois web service already caches requested whois records. However, these entries are only cached in memory. Stopping or restarting the service will flush this cache. Functionality will be added to save all entries in a database caching table, using a arbitrary expiration time. As a result, whois queries can be answered much faster. Again, scalability will be improved.

4.5 "Could Have's"

If all "Must Have's" are implemented, a more reliable, scalable and flexible system is created. Time permitting, we will implement "Could Have's"; new features. An interesting new feature might be the monitoring of the number of multi-homed networks.

5 Conclusion

5.1 Realization

During the four weeks of the project, we were able to realize most of the goals discussed in this document. We followed the steps listed in section 4.

We succesfully set up a development and deployment environment using Apache's Jakarta Tomcat, included with Sun's Java Web Services Development Pack. The necessary Apache Ant build script to build the web services was made, and developing and debugging the web services was done using Apple's XCode IDE, part of Apple's Developer Tools [8]. The parser was built using vim and the open source NetBeans IDE [16].

The functionality of the PHP/Ruby implementation of the web services was implemented in Java. The functionality of the software has been improved in three areas.

The first area of improvement is in the parser. The parser now handles data from multiple route servers. It compares the data and drops data from route servers that differ too much from the others. This improves the integrity of the BGP information in the database.

The second change is in the use of supporting database tables. These tables are updated by the parser software once a day. Where possible, the web services' functions merely retrieve the information from these tables, instead of using complex (and therefore time-consuming) queries on the main database table each time the web services' functions are called.

Finally, the parser for the whois information has improved. Instead of trying to parse every registry – the output formats of the various whois servers vary greatly – with one parser, different parser routines for each registry are used. This results in better and more complete *ASInfo* and *PrefixInfo* data structures being returned from the web services. Moreover, the whois information is now cached in two database tables, instead of in-memory.

5.2 Using Java

For reasons discussed in this document, the IPv6 web services have been coded in Java. Java source code usually compiles to byte code which runs in a virtual machine, the *Java Virtual Machine*, or JVM for short. JVMs exist for numerous operating systems, which allows *"compile once, run anywhere"*: Java byte code generated on one platform can be used on another platform. Unfortunately, while Java is often touted as being truly cross-platform, there are several problems. First, the Java Virtual Machines in existence all have their differences and they are not always compatible. Second, building web services with java requires infrastructure in the form of a build system and libraries. These libraries need to be present on every platform that should be able to run the project. However, because the Java libraries are designed to be very modular, this means that a large number of files need to be present on the target machine. Ensuring that all the necessary libraries are available or packaging the project so it includes them can be a hassle.

A second problem we encountered was that the web browsers used to test the Java client applet all behaved differently, and sometimes the java support seemed to be flawed. Refreshing applets to test changed functionality for example, didn't work on the Mac OS X platform. There was no time available to learn how to develop and debug Java applets with the XCode IDE for this platform.

5.3 Client implementation

As mentioned in the previous section, developing a Java client applet for the web services proved to be difficult. We really want to be able to demonstrate the web services with one or more clients, so we decided to adapt the existing Cocoa and PHP clients to use the new web services.

5.4 Multi-homed networks

We planned on showing, in some way, the number of multi-homed IPv6 networks. At the start of the project, we thought that the BGP data we stored contained the information necessary for detecting multi-homed networks. However, it became clear that our understanding of the way multi-homed networks are represented in the BGP table was not correct. Detecting multi-homed IPv6 networks would require more research on our part, but the limited time available for the project prevented us from doing so. We decided we would not, at this stage, implement this "could have".

5.5 Personal Notes

We would like to end this document with a personal note. Working on this project has been a pleasurable experience. It was nice to be able to continue the work that had already been done on the first implementation of the IPv6 web services. Hopefully, this project is just the beginning, not the end, of the development of a collection of web services and client that visualize and illustrate the IPv6 Internet. While we believe that we have created something useful in the four weeks of the project, we feel that many interesting areas relevant to this project could be researched and developed, given more time.

■

A Web Service Java Interface

The WSDL file specifying the web services interface is available on the project's web site. Since the WSDL file is not always easy to understand, we provide the Java interface to the web services' functions here as reference.

```
package nl.os3.ipv6ws;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Collection;

public interface IPv6ServiceIF extends Remote
{
    // WhoisService
    public PrefixInfo   getPrefixInfo(String prefix) throws RemoteException;
    public ASInfo       getASInfo(int asnr) throws RemoteException;
    public String       getWhoisInfoFromRegistry(String address,
                                String registry) throws RemoteException;

    // BGPInfoService
    public int getNumberOfEntriesForDate(String date) throws RemoteException;
    public int getNumberOfEntriesForDateMatchingPrefix(String date,
                                String prefixmatch) throws RemoteException;

    public SnapshotTotal[] getNumberOfEntriesBetweenDates(String firstDate,
                                    String secondDate)  throws RemoteException;
    public SnapshotTotal[] getNumberOfEntriesBetweenDatesMatchingPrefix(
                                    String firstDate, String secondDate,
                                    String prefixmatch) throws RemoteException;

    public BGP6Entry[] getDifferenceBetweenDates(
                                    String firstDate,
                                    String secondDate) hrows RemoteException;

    public String getEarliestDate() throws RemoteException;
    public String getLatestDate() throws RemoteException;
}
```

B Used Database Tables

B.1 table 6net_count

This table is used to store the number of entries in the BGP snapshots for a certain date.

```
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| date  | date    | YES  |     | NULL    |       |
| count | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

B.2 table 6net_main

This is the most important table of the database. The parsed BGP information is stored here.

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| prefix | varchar(40)  | YES  |     | NULL    |       |
| mask   | char(3)      | YES  |     | NULL    |       |
| asnr   | varchar(128) | YES  |     | NULL    |       |
| date   | date         | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

B.3 table 6net_whois_asinfo

This table functions as a cache for the whois web services. It holds the ASInfo data structures.

```
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| matchstring | varchar(255) | YES  |     | NULL    |       |
| autnum      | varchar(255) | YES  |     | NULL    |       |
| asname      | varchar(255) | YES  |     | NULL    |       |
| descr       | varchar(255) | YES  |     | NULL    |       |
| datum       | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

B.4 table 6net_whois_prefixinfo

This table functions as a cache for the whois web services. It holds the PrefixInfo data structures.

Field	Type	Null	Key	Default	Extra
matchstring	varchar(255)	YES		NULL	
inet6num	varchar(255)	YES		NULL	
netname	varchar(255)	YES		NULL	
descr	varchar(255)	YES		NULL	
country	varchar(255)	YES		NULL	
datum	varchar(255)	YES		NULL	

C Deliverables

A short list of files and documents that are part of the project:

- This document, containing background information about the project
- The deployed web services and related software (parser, WSDL file, database)
- The sources for the parser that retrieves BGP information from the route servers
- The sources for the web services
- The Cocoa/Objective-C client, including its sources
- The PHP client, including its sources

These items are all available from the project's web site: `http://pisa.toscana.os3.nl:8080/~jvb/6net/welcome.html`.

References

[1] "Peering," *WikiPedia, the free encyclopedia,* http://en.wikipedia.org/wiki/Peering; accessed January 30, 2004.

[2] Y. Rekhter, T. Li, *"A Border Gateway Protocol 4 (BGP-4),"* RFC 1771, Internet Engineering Task Force, March 1995, http://www.ietf.org/rfc/rfc1771.txt; accessed January 30, 2004.

[3] P. Marques, F. Dupont, *"Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing,"* RFC 2524, Internet Engineering Task Force, March 1999, http://www.ietf.org/rfc/rfc2545.txt; accessed January 30, 2004.

[4] Daniel Hilster, Andree Toonk, Aziz Ahrouch, Jeroen van Beek, Gert Jan Verhoog, *"IPv6 Growth Monitor Web Services,"* masters programme on system and network engineering, December 2003, http://pisa.toscana.os3.nl:8080/~gjv/; accessed January 30, 2004.

[5] Nakamura, Hiroshi, *"Release Notes - SOAP4R,"* http://rrr.jin.gr.jp/doc/soap4r/RELEASE_en.html

[6] Yukihiro Matsumoto, *"Ruby Home Page,"* http://www.ruby-lang.org/en/

[7] Apple Computer, *"Apple Developer - Cocoa,"* http://developer.apple.com/cocoa/

[8] Apple Computer, *"Apple Developer - XCode,"* http://developer.apple.com/tools/xcode/

[9] *"MySQL Home Page,"* http://www.mysql.com/

[10] *"PHP Home Page,"* http://www.php.net/

[11] *"Java Web Services Development Pack,"* Sun Microsystems, http://java.sun.com/webservices/

[12] *"Apache Ant,"* The Apache Ant Project, http://ant.apache.org/

[13] WebSphere Software Platform, *IBM,* http://www-306.ibm.com/software/info1/websphere/

[14] The Apache Jakarta Project, *"Apache Tomcat,"* http://jakarta.apache.org/tomcat/

[15] JBoss, *"The JBoss Application Server,"* http://www.jboss.org/

[16] NetBeans, *"The NetBeans IDE,"* http://www.netbeans.org/products/ide/