David James
Department of Computer Science
University of Toronto

HipVersion: Subversion for Hippos

Goal: To make it easier for Hippo developers to manipulate versioned repositories using Java

The goal of HipVersion is to make it easier for Hippo developers to manipulate versioned repositories using Java. This was accomplished by replacing Hippo's home-brewed Subversion Java bindings with the publicly available JavaHL bindings. The JavaHL bindings offer richer functionality, improved performance, proven reliability and reduced maintenance costs. This report describes the design process, history, and outcome of the HipVersion project.

Table of Contents

Chapter 1: Planning

1.1: Software Configuration Management and Hippo

Software configuration management (SCM) is the "tracking and control of software development" [1]. By providing facilities for "version control, concurrent development, release management, and change review", SCM tools make team programming more productive and less error-prone [2]. While SCM tools are heavily used in industry, few undergraduate courses make use of them because the overhead associated with setting them up and securing them is considerable [3]. The Hippo project aims to reduce this overhead by providing tools that will automatically configure the Subversion version-control system for academic use. Once Subversion is configured, Hippo will also allow both students and instructors to manage it via a simple web interface.

1.2: Goals

Before the HipVersion project was started, Hippo manipulated Subversion repositories using a home-brewed class named SvnProvider. The goal of the HipVersion project is to improve the functionality, performance and reliability of Hippo's Java bindings by replacing SvnProvider with the official Subversion Java bindings maintained by the Subversion team.

1.3: SvnProvider: The Old Subversion Integration Scheme

The SvnProvider class manipulated Subversion repositories by interacting with the Subversion command-line client using a primitive text-based interface. SvnProvider supported a very limited subset of Subversion functionality through the following three methods:

Figure 1: Functions Supported by the Old Subversion Integration Scheme

Method	Description
mkdir	Create a new directory
delete	Delete a directory or file
checkout	Retrieve files and/or directories

Each of the above methods built a text-based query and submitted it to the Subversion command-line client. If the command-line client crashed or returned an error code, then the method would throw an exception with the error message text.

The SvnProvider interface suffered from four major problems:

1. Limited functionality: SvnProvider does not support enough version control functionality to support Hippo. For example, SvnProvider did not support adding files to the Subversion repository.
2. Difficult maintenance: Every feature added to the SvnProvider class needed to be maintained by the Hippo team on an ongoing basis. Furthermore, if the team decides to support more advanced functionality, the SvnProvider class will grow in complexity and so will the maintenance obligations.
3. Uncertain Reliability: The SvnProvider class was only used for the Hippo project and was not subjected to rigorous tests. It is not known whether this code will perform reliably under more strenuous conditions.
4. Poor performance: The actual process of converting the query to text and interacting with the Subversion command-line client was CPU-intensive.

1.4: JavaHL: A Better Subversion Integration Scheme

There is no need for the Hippo project to be subjected to the limitations of SvnProvider because the Subversion team maintains a set of official Java bindings dubbed JavaHL. The JavaHL bindings make it easy to manipulate Subversion repositories using simple Java commands. The JavaHL bindings offer far superior functionality, performance, and reliability over our home-brewed implementation:

1. Full functionality: JavaHL offers complete access to the SCM functionality of Subversion.
2. Excellent Performance: JavaHL communicates directly with the Subversion server using the Subversion client library.
3. Proven Reliability: The JavaHL team maintains a large test suite that covers the majority of its documented functionality. Furthermore, JavaHL is used widely within the Subversion community and undergoes regular maintenance updates and fixes.

Because the official Java bindings are maintained by the Subversion developers, these advantages come with no additional maintenance obligations. In fact, since these bindings replace the SvnProvider interface, switching to JavaHL will actually *reduce* the amount of code that Hippo developers will need to maintain.

1.5: Migration Issues

If SvnProvider is replaced with JavaHL, the Hippo developers will need to compile and install JavaHL. Unfortunately, JavaHL suffers from several bugs that make it confusing, difficult and slow to install:

1. Broken Download: At the start of our project, the latest stable Subversion did not contain a working version of JavaHL. The latest unstable version of Subversion contains fixes that are necessary to compile JavaHL [4].
2. Need to Use Subversion to Download JavaHL: While stable versions of Subversion are available on the Subversion website, the unstable version is not. The only way to get the unstable Subversion source code is to download it using Subversion. Therefore a new user must first download and compile the stable Subversion distribution before downloading the unstable version of Subversion [4].
3. Non-standard Installation Targets: Once you have configured JavaHL to install, you will need to remember to run 'make javahl' and 'make install-javahl' to compile and install it. The standard targets 'all' and 'install' will only install the base Subversion package [5].
4. Broken Installation Dependencies: The installer crashes if the user attempts to install JavaHL without installing the latest version of the base Subversion package [6]. If the user already has the stable version of Subversion installed and decides not to replace it, JavaHL may crash unexpectedly due to version incompatibilities [7].
5. Missing Directories: The compilation step of JavaHL will fail the first time it is run due to a missing directory. To work around this problem, the user must create the missing directory [4, 8].
6. Test Suite Fails Due to Typo: Due to a minor typographical error in the testing code, JavaHL does not pass its test suite [9].
7. Can't Test Before Installation: Currently, JavaHL must be installed first before it can be tested. This restriction is undocumented and users are often surprised when their JavaHL build fails tests for no obvious reason [10].

8. Slow Compilation: The Java header files and classes are always recompiled, even if they already exist. This behaviour is suboptimal and slows the installation process [11].

If they are not properly managed, these migration issues could create roadblocks. Forcing each member to debug minor issues in the JavaHL installer could tie up the entire Hippo team. Furthermore, forcing these installation issues on users could create a bad first impression. Therefore it is clear that JavaHL's installation process needs to be improved.

1.6: Easy Installation for Casual Users

Eclipse is an integrated development environment for Java, and Subclipse is a plug-in that integrates Subversion with Eclipse. Subclipse uses JavaHL to manipulate Subversion repositories. Because the JavaHL bindings can sometimes be difficult to set up, the Subclipse team devised a simple adapter called SVNClientAdapter that would only make use of JavaHL if it is available. If not, SVNClientAdapter will send requests using the Subversion command-line client. This adapter offers the full functionality of Subversion to all users regardless of whether they have JavaHL installed.

This flexible approach offers all of the benefits of JavaHL to users who are willing to install it. If the users need speed, they should install JavaHL and SVNClientAdapter will use JavaHL to achieve optimal performance. If speed is not a major concern, then the user does not need to install JavaHL; SVNClientAdapter will work just as well without it, even if it runs a bit slower.

1.7: Easy Installation for Power Users

While the flexible approach offered by SVNClientAdapter helps casual users, it doesn't solve the installation hassles for users who need the speedy performance of JavaHL. To make JavaHL easier to install, the bugs in the installation process need to be fixed. The table below outlines each installation issue and the plan to fix it:

Figure 2: JavaHL Installation Problems and the Required Changes

Problem	Required Changes to Make JavaHL Easy to Install
Broken Download / Need to use Subversion to download JavaHL	None. The JavaHL download will be fixed in the next stable version of Subversion.
Non-standard Installation Targets	Upgrade JavaHL to use standard installation targets
Broken Installation Dependencies	Upgrade JavaHL to automatically build and install these dependencies
Missing Directories	Upgrade JavaHL to automatically create missing directories
Test Suite Fails Due to Typo	Fix the typographical error
Can't Test Before Install	Upgrade JavaHL to support testing before install
Slow Compilation	Upgrade JavaHL to only recompile when necessary

Figure 2 identifies six areas where the installation process of JavaHL needs to be improved. Fortunately, the Subversion team is a friendly group of people and they graciously accept beneficial fixes. If these changes are accepted by the Subversion team, the Subversion team will handle all of the maintenance obligations. Therefore the Hippo team will not need to continually upgrade our fixes to work with the latest version of Subversion. This will save the Hippo team substantial effort on an ongoing basis.

1.8: Ant Integration

The Hippo build script is written in XML and is parsed by a tool called Ant. Hippo uses a custom Ant task called SvnTask to manipulate the Subversion repository from within the build script. In order to upgrade Hippo to use SVNClientAdapter instead of SvnProvider, it will be necessary to upgrade Hippo's custom-made Ant task to support the new features of SVNClientAdapter. Fortunately, this task is easy: the Subclipse team has already created an Ant task which supports the full functionality of SVNClientAdapter. By switching to the Subclipse Ant task, the Hippo team will take advantage of the superior functionality, performance and reliability of SVNClientAdapter. This switch will also reduce the amount of code that Hippo developers will need to maintain.

1.9: The HipVersion Integration Plan

The plan is to replace Hippo's custom-made SvnProvider class with SVNClientAdapter. Because SVNClientAdapter offers more functionality than SvnProvider and has no external dependencies, it is expected that the transition to the new system will be smooth.

The following table describes the changes required to support the switch to SVNClientAdapter.

Figure 3: Required Changes to Switch to SVNClientAdapter

Situation	Required Change
Initialize Hippo: When the Hippo database is first initialized, Hippo creates a directory in the Subversion repository to contain Hippo-related files.	Upgrade initializeHippoTask class to use new SVNClientAdapter
Create Project: When a user creates a new project, Hippo creates a directory in the Subversion repository to contain the files for that project.	Upgrade addProject function to use new SVNClientAdapter
Delete Project: When a user deletes a project, the associated directory in the Subversion repository is also removed.	Upgrade removeProject function to use new SVNClientAdapter
Delete Hippo: The Hippo build script uses a custom Ant task called SvnTask to delete Hippo-related files in the Subversion repository.	Upgrade build script to use the new SvnTask class created by the Subclipse team

1.10: Tasks

The following list identifies the tasks required to complete the objectives and the estimated time requirements for each task. The total estimated time for the project was 152 hours.

1. Ensure that JavaHL passes all tests (Section Total: 9 hours)
    - Determine the cause of the failure in the test suite (2 hours)
    - Build patch to fix error in test suite (1 hour)
    - Submit patch to Subversion developers and respond to feedback
        - Submit patch (1 hour)
        - Respond to feedback (2 hours)
        - Resubmit patch with changes if necessary (3 hours)

2. Make JavaHL easy to test (Section Total: 12 hours)
    - Allow JavaHL to be tested before installation
        - File bug report (2 hours)
        - Build patch [in collaboration with Holger Thon] (4 hours)
        - Submit patch to Subversion developers and respond to feedback
            - Submit patch (1 hour)

        - Respond to feedback (2 hours)
        - Resubmit patch with changes if necessary (3 hours)

3. <u>Make JavaHL easy to build</u> (Section Total: 40 hours)
    - Fix issue 2032: JavaHL build fails due to missing directory
        - Determine the cause of the 'missing directory' problems (6 hours)
        - Build patch (2 hours)
        - Submit patch to Subversion developers and respond to feedback
            - Submit patch (1 hour)
            - Respond to feedback (2 hours)
            - Resubmit patch with changes if necessary (3 hours)
    - Fix javahl-lib compilation dependencies
        - Research problem (1 hour)
        - Build patch (1 hour)
        - Submit patch to Subversion developers and respond to feedback
            - Submit patch (1 hour)
            - Respond to feedback (2 hours)
            - Resubmit patch with changes if necessary (3 hours)
    - Fix javahl-java and javahl-javah compilation dependencies
        - Determine how to fix the compilation dependency problems with JavaHL (4 hours)
        - Build patch (8 hours)
        - Submit patch to Subversion developers and respond to feedback
            - Submit patch (1 hour)
            - Respond to feedback (2 hours)
            - Resubmit patch with changes if necessary (3 hours)

4. <u>Make JavaHL easy to install</u> (Section Total: 20 hours)
    - Fix JavaHL installation dependencies
        - Determine how to fix the installation dependency problems with JavaHL
            - ... by corresponding with Subversion developers (4 hours)
            - ... by debugging the code myself (6 hours)
        - Build patch (4 hours)
        - Submit patch to Subversion developers and respond to feedback
            - Submit patch (1 hour)
            - Respond to feedback (2 hours)
            - Resubmit patch with changes if necessary (3 hours)

5. <u>Write JavaHL tutorial on getting a simple JavaHL example up and running</u> (Section Total: 8 hours)
    - Document list of commands required to compile and install Subversion and JavaHL on Linux (3 hours)
    - Direct Mac OS X and Windows users to JavaHL binaries on Subversion website (1 hour)
    - Write simple JavaHL example (1 hour)
    - Explain simple JavaHL example in HTML format (1 hour)
    - Send document to Subversion developers' list and reply to questions (1 hour)

- Send document to Neon developers' list and reply to questions (1 hour)

6. Convert Hippo to use ISVNClientAdapter (Section Total: 9 hours)
    - Add SVNClientAdapter and JavaHL into Hippo classpath (3 hours)
    - Update ca.utoronto.hippo.model.Hippo.createSvnProvider function to create an ISVNClientAdapter object (2 hours)
    - Update ca.utoronto.hippo.model.Hippo.addProject function to use ISVNClientAdapter to create a directory (1 hour)
    - Update ca.utoronto.hippo.model.Hippo.removeProject function to use ISVNClientAdapter to delete a directory (1 hour)
    - Convert ca.utoronto.ant.InitializeHippoTask to use ISVNClientAdapter to make a directory (1 hour)
    - Create ca.utoronto.hippo.model.Hippo.repositoryPathAsURL function to convert a path in the working copy to a URL in the repository (1 hour)

7. Administrative tasks (Section Total: 54 hours):
    - Setup
        - Install and configure JavaHL and its prerequisites (3 hours)
        - Set up distributed repository using SVK to track my changes to the Subversion repository (2 hours)
        - Learn proper procedures for submitting Subversion patches
            - Read Subversion documentation on how to submit patches (4 hours)
            - Correspond with Subversion developers to clarify procedures (3 hours)
        - General Proposal to Subversion Developers
            - Write initial post (3 hours)
            - Respond to questions (3 hours)
    - Assist team with JavaHL issues when necessary (16 hours)
    - Write final report detailing our progress this term (20 hours)

Chapter 2: Implementation and Evaluation

2.1: Implementing the HipVersion Plan

The plan for switching to SVNClientAdapter was implemented in the 577th revision of Hippo. While this change enabled Hippo developers to access the full functionality of Subversion, it actually simplified the Hippo code: 168 lines of code were deleted but only 63 new lines were added.

- <u>New Java Libraries</u>
  - svnClientAdapter.jar: Contains the SVNClientAdapter class.
  - svnAnt.jar: Contains the Subclipse SvnTask class
- <u>Modified files</u>
  - src/ca/utoronto/hippo/model/Hippo.java
    - Created <u>repositoryPathAsURL</u> function to convert a path on the local disk to a URL in a Subversion repository
    - Upgraded <u>addProject</u> method to use new SVNClientAdapter
    - Upgraded <u>removeProject</u> method to use new SVNClientAdapter
    - Upgraded <u>createSVNProvider</u> method to create a new SVNClientAdapter object. This function will create a JavaHL object if possible. Otherwise, it will use the command line client.
  - src/ca/utoronto/ant/InitializeHippoTask.java
    - Converted <u>execute</u> method to use SVNClientAdapter
  - build.xml
    - Replaced custom SvnTask in <u>clean_svn</u> target with Subclipse SvnTask class
- <u>Deleted classes</u>:
  - ca.utoronto.hippo.vcs.TestSvnProvider
  - ca.utoronto.hippo.vcs.SvnProvider
  - ca.utoronto.hippo.ant.SvnTask

2.2: Making JavaHL Easy to Install

A three-step process was used to implement the plan set out in Section 1.7 for making JavaHL easy to install:

1. Submit a proposal to the Subversion developers explaining the purpose of each change and how it could be implemented
2. If the proposal was accepted, create and submit a patch
3. If the patch was accepted, nominate it for inclusion in an appropriate, stable version of Subversion

The secret to success in submitting a patch is to be polite yet persistent: every time a developer raises a concern with your patch, fix it and resubmit. If you get no response, wait a month and then e-mail a reminder to the list. Eventually your patch will either be accepted or rejected; either way, you'll know how you can improve for next time.

The success rate of my attempts at solving the four problems is detailed below. 80% of my proposals and 80% of my submitted patches were eventually accepted. I expect that my remaining patch and proposal will be accepted during the next few months. Considering that the Subversion developers are very selective in which patches they accept, the 80% acceptance rate is a mark of success.

Figure 4: Proposals and Patches for Making JavaHL Easy to Install

Problem	**Proposal**		**Patch**		**Included in version**
	Submit	**Accept**	**Submit**	**Accept**	
Non-standard Install Targets	☑	☐	N/A	N/A	N/A
Broken Install Dependencies	☑	☑	☑	☑	1.2.0
Missing Directories	☑	☑	☑	☑	1.1.2
Test Suite Fails Due to Typo	☑	☑	☑	☑	1.1.0
Can't Test Before Install	☑	☑	☑	☐	N/A
Slow Compilation	☑	☑	☑	☑	1.2.0

The content of each patch and proposal is described below.

2.2.1: Non-standard Install Targets

Problem: Once you have configured JavaHL to install, you will need to remember to run 'make javahl' and 'make install-javahl' to compile and install it. The standard targets 'all' and 'install' will only install the base Subversion package.

Solution: After I raised this issue, Justin Erenkrantz said that he prefers the status quo because he doesn't want the Java bindings to behave differently from those of other languages [12, 13]. On the advice of Max Bowsher, I reposted my question as a general question about streamlining binding installation for any language [14, 15]. In the new topic, Erik Huelsmann expressed modest support for the idea, but Ben Reser noted that he had tried this idea before and received a lot of complaints [16, 17]. While I still hope that I can eventually convince the Subversion developers to switch to standard target names, I decided to postpone implementing this idea.

2.2.2: Broken Install Dependencies

Problem: The installer crashes if the user attempts to install JavaHL without installing the latest version of the base Subversion package.

Solution: The first time I tried to install JavaHL, the install script crashed and exited with a weird error. Searching for a way to work around the bug, I tried installing Subversion first before installing JavaHL, and the installation worked. Nevertheless, I found this bug disconcerting. This bug could easily trip up new Hippo users.

After many hours of debugging, I determined that this error occurs because Subversion does not keep track of the extra installation dependencies for all of its libraries. Fixing this bug involved upgrading the build script to keep track of these dependencies. After a round of review, I submitted my patch to the Subversion list [18]. Greg Hudson committed this patch in r11050 [19]. After I nominated this patch to be included in Subversion 1.1.2, it was approved by Greg Hudson and Justin Erenkrantz. With the approval of one more committer, this patch will be accepted into Subversion 1.1.2.

2.2.3: Missing Directories

Problem: With some build configurations, the compilation step of JavaHL will fail the first time it is run due to a missing directory.

Solution: The easy solution to this problem is simply to create the missing directory in the Makefile, as attempted by Holger Thon [20]. The Subversion developers, however, asked me to look deeper into the true cause of the problem before so quickly prescribing a fix. Looking more closely, I discovered that the missing directory is supposed to be created by a Makefile target called *mkdir-init*; however, the *mkdir-init* target is not always executed. By switching Subversion to always run *mkdir-init* in the configuration script, I solved this problem. My final patch was committed in revision r11047 by Greg Hudson [21, 22]. After I nominated this patch for inclusion in Subversion 1.1.2, it was committed by Erik Huelsmann and approved by Greg Hudson, Max Bowsher, and Daniel Rall [6].

2.2.4: Test Suite Fails Due to Typo

Problem: Due to a minor typographical error in the testing code, JavaHL does not pass its test suite.

Solution: I fixed the typographical error and posted a patch to the mailing list [9]. Patrick Mayweg committed my patch in revision 10773 [50]. McClain Looney found this patch quite important and nominated it for inclusion in Subversion 1.1.0. Max Bowsher backported my patch to version 1.1.0 of Subversion just one week before its release date with the approval of Patrick Mayweg, Greg Hudson, Karl Fogel, and Ben Reser [23].

2.2.5: Can't Test Before Install

Problem: Currently, JavaHL must be installed first before it can be tested. This restriction is undocumented and users are often surprised when their JavaHL build fails tests for no obvious reason.

Solution: When I first compiled JavaHL, I was surprised to see the tests fail. It is usually not a good idea to install a piece of software unless its tests pass. Nevertheless, I installed it anyway, and discovered that the tests only work *after* you have installed the software. After C. Michael Pilato confirmed my problem and noted that this issue also prevents him from using the bindings in the main Subversion test suite, I posted this problem to the issue tracker as issue 2040 [24, 25, 26, 10].

Soon after I posted the issue, Holger Thon posted a simple solution to the Issue Tracker that set the Java library path to point to the Libtool temporary build directory for the Java libraries [10]. To ensure that Holger's patch received the attention of the Subversion developers, I posted his patch to the Subversion developers' mailing list with a few small changes to make it more portable. In response, Greg Hudson asked whether there was a better way to do this than messing with temporary build directories [27].

Holger Thon responded to this feedback by trying a new trick which would request the location of the temporary directory from Libtool by using the LD_LIBRARY_PATH variable [28]. After I pointed out that this technique does not work on all platforms, he followed Ben Reser's advice and used the Apache Portable Runtime to achieve better portability [29, 30, 31]. A few weeks later, I looked over the patch again and reposted it with a Subversion-style log message and a few minor improvements [32]. A few months later, at the urging of Karl Fogel, Justin Erenkrantz took a look at this patch and noted that he probably won't get around to it for a long time because he dislikes Libtool [33, 34].

Looking at the patch again, I decided to take another look at Holger's original method of simply accessing the Libtool temporary build directory. Looking at the Libtool documentation, I discovered that Libtool exports a variable called "$objdir" which specifies the name of the directory where the compiled versions of the library are stored [35]. Therefore there is no need to include complex code to extract this directory from Libtool; I can simply use $objdir [36]. With this change, the patch was much simpler. No feedback has been received yet on this new patch [37].

2.2.6: Slow Compilation

Problem: The Java header files and classes are always recompiled, even if they already exist. This behaviour is suboptimal and slows the installation process.

Solution: After issue was first pointed out by Justin Erenkrantz, I posted it to the issue tracker as issue 2039 [13, 38]. Reading up on the issue, I soon discovered that each Java file could potentially depend on any or all of the other Java files. While dependency checkers for Java do exist, it's definitely easier and often faster to simply ignore dependencies and just recompile everything if any of the files change.

Therefore I decided to handle dependencies in a very conservative way: I assume that every Java file depends on every other Java file. I regenerate the full set of class files any time that any of the Java files changes. This solution prevents unnecessary recompiles when no files have been changed. Both Holger Thon and I posted fixes; Holger Thon recommended that the developers use my fix because his patch was incomplete [38]. Justin Erenkrantz committed my patch in r12018 [39].

2.3: Measuring Ease of Installation: Before and After

At the outset of the HipVersion project, installing the JavaHL bindings was a complicated twelve step process [4]:

1. Download Subversion 1.06
2. Compile Subversion 1.06
3. Install Subversion 1.06
4. Use Subversion 1.06 to download the latest version of Subversion
5. Copy the Apache Portable Runtime directory from the Subversion 1.06 directory to the newly downloaded Subversion directory
6. Generate the *Configure* script using *autogen.sh*
7. Generate the *Makefile* using the *Configure* script
8. Build Subversion
9. Install Subversion
10. Create the *subversion/bindings/java/javahl/classes* directory
11. Build JavaHL
12. Install JavaHL

Thanks in part to my contributions, the JavaHL installation process is much easier with Subversion 1.2.0:

1. Download Subversion 1.2.0
2. Generate the Makefile using the Configure script
3. Install JavaHL by typing *make install-javahl*. All dependencies will be built and installed automatically.

2.4: Sample Code

The following code sets up a new ‘SVNClientAdapter’ object by asking SVNClientAdapterFactory for the best available implementation. Currently, this code chooses only between a JavaHL and a command-line implementation of SVNClientAdapter. In the next version of SVNClientAdapter, there will also be a pure Java implementation of SVNClientAdapter. Because this code asks for the “best available” implementation, it does not exclude the pure Java implementation. When the new version of SVNClientAdapter is released, this code will support using pure Java to interact with Subversion.

```
public SVNClientAdapterWrapper() throws SVNClientException {
    fSvnClient = SVNClientAdapterFactory
            .createSVNClient(SVNClientAdapterFactory
                    .getBestSVNClientType());
}
```

The following code demonstrates how to create a new directory or file directly in the Subversion repository.

```
/* Create a new directory directly in the Hippo Subversion repository */
public void mkdir(String path, String message) throws SVNClientException {
    fSvnClient.mkdir(repositoryPathAsURL(path), message);
}

/* Create a new file directly in the Hippo Subversion repository */
public void addFile(String path, String message) throws SVNClientException {
    fSvnClient.copy(path, repositoryPathAsURL(path), message);
}

/* The repositoryPathAsURL function, defined in SVNClientAdapterWrapper,
   converts the specified path to a URL in the Hippo repository */
```

2.5: Feature Matrix

As demonstrated below, JavaHL provides significantly better coverage of the Subversion feature set than our old system; JavaHL supports almost all of the Subversion features.

Figure 5: Subversion Feature Matrix with our old and new integration schemes [40]

Feature	Descriptions	Old	New
add	Schedule a file or directory for addition to the repository	☐	☑
blame	Show author and revision information in-line	☐	☑
cat	Output the contents of the specified files or URLs	☑	☑
checkout	Check out a working copy from a repository	☑	☑
cleanup	Recursively clean up the working copy	☐	☐
commit	Send changes from your working copy to the repository	☐	☑
copy	Copy a file or directory in a working copy or in the repository	☐	☑
delete	Delete an item from a working copy or the repository	☑	☑
diff	Display the differences between two paths	☐	☑
export	Export a clean directory tree	☐	☑
import	Recursively commit a copy of PATH to URL	☐	☑
info	Print information about PATHs	☐	☑
list	List directory entries in the repository	☐	☑
merge	Apply the differences between two sources to a local path	☐	☐
mkdir	Create a new directory under version control	☑	☑
move	Move a file or directory	☐	☑
propdel	Remove a property from an item	☐	☑
propedit	Edit the property of one or more items under version control	☐	☐
propget	Print the value of a property	☐	☑
proplist	List all properties	☐	☑
propset	Set PROPNAME to PROPVAL on files, directories, or revisions	☐	☑
resolved	Remove 'conflicted' state on working copy files or directories	☐	☑
revert	Undo all local edits	☐	☑
status	Print the status of working copy files and directories	☐	☑
switch	Update working copy to a different URL	☐	☐
update	Update your working copy	☐	☑

Descriptions from *Version Control with Subversion* by Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato [40]. Licensed under the Creative Commons Attribution license [41].

2.6: Test Coverage

A sizable majority (68%) of the JavaHL library is covered by the JavaHL test suite, as demonstrated below. Each test function builds a set of files on disk, commits it to a repository, and tests whether a specific command behaves as desired.

Figure 6: Functions that implement and test each JavaHL feature

Feature	Function	Test Functions
add	addDirectory, addFile	testBasicAddIgnores
blame	blame	
cat	getContent	testBasicCat
checkout	checkout	testBasicCheckout, testBasicCheckoutDeleted
commit	commit	testBasicCommit
copy	copy	
delete	remove	testBasicDelete
diff	diff	
export	doExport	
import	doImport	testBasicImport, testBasicImportIgnores
info	getDirEntry	testBasicInfo
list	getList	testBasicLs
mkdir	mkdir	testBasicMkdirUrl
move	move	
propdel	propertyDel	
propget	propertyGet	testBasicDelete
proplist	getProperties	
propset	propertySet	testBasicDelete
resolved	resolved	testBasicConflict
revert	revert	testBasicRevert
status	getSingleStatus, getStatus	testBasicStatus
update	update	testBasicUpdate, testBasicMergingUpdate, testBasicConflict

JavaHL implements its features by calling the Subversion Client Library. The Subversion Client Library, being the main library used by almost all of the Subversion clients, is tested even more thoroughly than JavaHL.

2.7: Task Checklist

The following table lists the major tasks established in our plan and outlines whether they were completed. The HipVersion team has accomplished all of its objectives.

Task	**Estimated Hours**	**Completed?**
Ensure that JavaHL passes all tests	9	☑
Make JavaHL easy to test	12	☑
Make JavaHL easy to build	40	☑
Make JavaHL easy to install	20	☑
Write JavaHL tutorial on setting up a simple JavaHL example	8	☑
Convert Hippo to use ISVNClientAdapter	9	☑
Administrative tasks	54	☑

Chapter 3: Discussion

By participating in the Hippo project and contributing patches to the Subversion project, I learned quite a bit about how to approach this type of project. In the hopes that these lessons might be useful to future developers who contribute to open source projects, I have summarized some of the lessons below.

3.1: Lessons for Contributing to Hippo

<u>Start early</u>

I started this project early, in August, and finished the bulk of the required work by the end of September. It was nice to finish my requirements early on the term so that I could focus on other courses when necessary.

<u>Pick reasonable goals</u>

Project courses can be very demanding. It is important to allocate time for your other courses so that the project course does not take up all of your time. To ensure that this will be possible, you must pick goals at the outset of the project that you can complete in the time you have available.

<u>External tools save development and maintenance time</u>

Developing a tool in-house takes time and will cost future developers of the project time to maintain on an ongoing basis. By selecting an external tool instead, I saved the Hippo team a substantial amount of time. With JavaHL, I was able to implement more features much more quickly than I would have otherwise expected.

<u>Contributing to open source helps Hippo</u>

I contributed several fixes to the JavaHL project to make it easier to install. It would have been easier for me to simply keep our contributions in-house because I would not have had to meet the Subversion project's strict code-review policies. However, by contributing to the Subversion project, I learned how to improve the quality of my code while saving future Hippo developers from needing to maintain my fixes to JavaHL.

3.2: Lessons for Contributing to Subversion

Don't just read the documentation. Summarize the key points

The first step in contributing to any Open Source project is to read the documentation. Before I contributed to the Subversion project, I read the 'HACKING' file, which provides advice on how to submit patches to the Subversion developers. When I later prepared a presentation on how contribute to open source, I summarized the key points of the 'HACKING' file on slides, and I began to understand the documentation in a much deeper way than I had ever before. If I had taken this step earlier, it would have improved the quality of my code.

Split patches into manageable portions

Early in the project, I made the mistake of submitting a huge patch to fix all of the installation problems with JavaHL [42]. This patch was rejected because it was too difficult for any one person to review because it contained so many issues. If it is possible, it's better to submit a few small patches instead. Small patches are easier to review and are more likely to be integrated.

Comment your code to prevent confusion

The developer who commits your code to the Subversion project will need to understand every single line of your code. While excessive comments may detract from understanding, a few key comments here and there can drastically reduce the amount of time the reviewer will need to spend reading your code. This will result in your patch being accepted sooner.

Write detailed log messages

The log message for your patch should contain everything that developers will need to both understand your code and decide whether you have selected the best method for achieving your goal. It's a good idea to outline a few alternative ways of achieving your goal, and explain why you didn't choose them. By exposing your logic, you reduce the amount of time that the reviewer will need to spend deciding whether to accept your patch.

Let your patches sit for a few days before submitting them

No matter how carefully I check over a patch, I will occasionally realise the next day that I had forgotten something. When I wait an extra day before submitting the patch, the quality of my code improves.

Admit Inexperience

If you're new to an open source project, be sure to say so when you submit your patch. It's quite likely that your first few patches will have problems, and it's better to admit it up front than to be chastised for it later. If you admit that you're a beginner and that you're looking for advice, the developers will be more patient and will take the time to explain how you can improve.

Be Patient and Communicate

The task of carefully triple checking your patch to ensure that its entirely correct takes time. Checking the correctness of a patch involves much more than simply reading over each line of code that you submit; it also means comparing your code to the rest of the Subversion code to ensure your code fits in. You'll need to regularly e-mail the developers to ensure you're attacking the problem using the right method and that you're following the appropriate conventions. This process is slow, but it works: it produces the highest quality software.

Chapter 4: Conclusion

The goal of HipVersion is to make it easier for Hippo developers to manipulate versioned repositories using Java. This was accomplished by replacing Hippo's home-brewed Subversion Java bindings with the publicly available JavaHL bindings. The JavaHL bindings offer richer functionality, improved performance, proven reliability and reduced maintenance costs. Hippo is now leaner and meaner: it has less code but more features.

At the outset of the HipVersion project, JavaHL was very difficult to install. With the assistance and support of the Subversion developers, I have fixed these flaws: JavaHL is now easy to install. By submitting my fixes to the Subversion project, I have freed future Hippo developers from the obligation of maintaining my patches.

In the future, Hippo developers may wish to upgrade Hippo to function using pure Java without any need for either JavaHL or the Subversion command-line client. Fortunately, the Subclipse developers plan to include support for this feature in the next stable version of the SVNClientAdapter library. When the new version of SVNClientAdapter is released, Hippo developers can upgrade and start using the pure Java library immediately: no code changes will be necessary.

Appendix A: Glossary

A1: Terminology

Makefile: A script that builds a program

Log Message: A message that describes a patch

Patch: A list of changes (usually to source code)

Version Control System: A program that tracks the versioned history of a set of files

Working Copy: An "ordinary directory tree on your local system, containing a collection of files" [40].

A2: Software

Ant: A Java-based tool for building programs [43]

Make: A Unix-based tool for building programs [44]

Hippo: A lightweight framework supporting undergraduate programming projects [45]

Libtool: A tool for building shared libraries in a portable manner [46]

JavaHL: The official Subversion Java Bindings supported by the Subversion development team

Subversion: A popular version control system [47]

Subversion Java Bindings: A Java library for manipulating Subversion repositories

Subversion Client Library: A C library for manipulating Subversion repositories

Subversion Command-line Client: A command-line program that can manipulate Subversion repositories

SVNClientAdapter: A publicly available library that can manipulate Subversion repositories using either **JavaHL** or the **Subversion Command-line Client** [48]

SvnProvider: Hippo's home-brewed class for manipulating Subversion repositories

Appendix B: JavaHL Proposals

RFC 1: Making JavaHL easier to maintain? [12]

```
Subversion developers,

I'm planning to work on making JavaHL easier to maintain. What would
you like me to work on?

Here are some of my ideas (can you give me some feedback?):
- JavaHL should be integrated into the standard build process
-- If --enable-javahl is set, JavaHL should be built, tested, and
installed when you call 'make', 'make check', and 'make install'
-- We should set up a test script to build the latest version of
JavaHL and e-mail us when it doesn't pass tests
- We should autogenerate the JNI headers for JavaHL instead of
updating them manually
-- Patrick Mayweg has an Ant script to autogenerate the JNI headers.
We should call this script when users make javahl.
- We should make JavaHL easier to test
-- make 'check-javahl' should automatically build the test suite if it
has not yet been built
-- make 'check-javahl' should work even if javahl has not yet been installed
```

RFC 2: Streamlining Binding Installation [15]

```
PROBLEM:

Currently, the process of building and installing bindings is quite complex.

For example, here is how to build and test Subversion and the JavaHL
bindings (from a source tarball):
1. ./configure [with some options]
2. make
3. make install
4. make check
5. make javahl
6. make javahl-tests
7. make install-javahl
8. make check-javahl

The process of building and testing the SWIG bindings is also complex
-- and it requires an entirely different set of commands from those
required to build JavaHL.

PROPOSED SOLUTION:

If the user asks configure to build a binding, build it and test it
with the rest of the Subversion.

The build process (for all bindings) will now be:
1. ./configure [with some options]
2. make
```

```
3. make install
4. make check

PROS
- The new process is simpler than the old process (half as many lines!)
- All bindings can be configured and built using the same process
(only difference is a configure option!)
- I generally expect an application to fully install after I type
"./configure [with some options] && make && make install". Subversion
will now operate as I expect.
- Other advantages?

CONS
- The new process is different from the old process (Users will have
to learn the new process)
- Other disadvantages?

Comments?
```

RFC 3: Best way to get "objdir" variable from libtool? [36]

```
According to the LIBTOOL documentation, objdir is "The name of the
directory that contains temporary libtool files". What do you think is
the best way to get the value of the "objdir" configuration variable
from libtool inside configure.in?

Currently, Subversion includes the libtool macros directly inside
configure.in using sinclude(ac-helpers/libtool.m4)

The libtool.m4 script sets the $objdir variable. It appears from my
reading of the LIBTOOL documentation that "objdir" is a documented
output of this libtool.m4 script and it is therefore safe for us to
directly use the $objdir variable that is set by libtool.m4. Am I
correct?

See the relevant section of the LIBTOOL documentation at:
  http://www.delorie.com/gnu/docs/libtool/libtool_71.html

Here's an example snippet of code using this variable:
FIX_JAVAHL_LIB="ln -sf libsvnjavahl-1.dylib \`dirname
\$(libsvnjavahl_FILENAME)\`/$objdir/libsvnjavahl-1.jnilib"

Cheers,

David
```

RFC 4: If bindings are enabled, install them with standard 'install' target [5]

```
MY PROPOSAL:
If the user chooses to enable a set of bindings using "configure",
then we should compile and install them with the standard targets. For
example, if the user types "./configure --enable-javahl=yes && make &&
make install", then we should include JavaHL with the Subversion
installation.

JUSTIFICATION:
- If users choose to enable JavaHL, then clearly they want to compile
and install it. If they don't want to install JavaHL, then they
wouldn't have enabled it.
- For most programs, the standard "all" and "install" targets compile
and install the entire program. If users wish to enable or disable
specific sections of a program, then they can do so by specifying
different options when they call configure. See the official GNU
coding standards for Makefiles at
<http://www.gnu.org/prep/standards/html_node/Standard-Targets.html>.

If the Subversion team is in favour of my proposal, then I will submit
a patch to implement it. Please let me know :)

Cheers,

David

P.S. Related threads:
- [RFC] Streamlining Binding Installation
http://svn.haxx.se/dev/archive-2004-09/0075.shtml
- [SVN RFC] Making JavaHL easier to maintain?
http://svn.haxx.se/dev/archive-2004-09/0054.shtml
```

Appendix C: Log Messages for JavaHL Patches

C1. Broken Install Dependencies

Patch committed in r11050 by Greg Hudson [19]

```
Add install dependencies for libsvn_fs and libsvn_fs_base.  From David
James <james@cs.toronto.edu>.

* configure.in
  (libsvn_fs, libsvn_ra): Added SVN_RA_LIB_INSTALL_DEPS and
    SVN_FS_LIB_INSTALL_DEPS to keep track of installation dependencies
    for libsvn_ra and libsvn_fs

* Makefile.in
  (libsvn_fs, libsvn_ra): Added SVN_RA_LIB_INSTALL_DEPS and
    SVN_FS_LIB_INSTALL_DEPS to keep track of installation dependencies
    for libsvn_ra and libsvn_fs

* build.conf:
  (libsvn_fs): Added $(SVN_FS_LIB_INSTALL_DEPS) to install deps of
    libsvn_fs
  (libsvn_ra): Added $(SVN_RA_LIB_INSTALL_DEPS) to install deps of
    libsvn_fs_base

* build/generator/gen_base.py
  (gen_base.Target): Added 'add-install-deps' option for build.conf to
    list additional installation dependencies

* build/generator/gen_make.py
  (gen_make.Generator.write): Upgraded dependency checker to support
    manual cross-library installation dependencies
```

Patch committed in r12016 by Justin Erenkrantz [49]

```
Resolve Issue #2102 by adding the right dependencies for javahl bindings.

* build.conf
 (javahl-javah): Add $(javahl_java_DEPS) to the dependencies for javahl-javah
 so that the javahl-javah target will work independently.
 (libsvnjavahl): Add $(javahl_javah_DEPS) and $(javahl_java_DEPS) to the
 dependencies for libsvnjavahl so that the libsvnjavahl target will work
 independently.

Note that this is different than what is in the issue tracker as adding
the simple targets is incorrect as make will assume that 'javahl-java'
and 'javahl-javah' must always be rebuilt -- instead, use their dependencies.
(This follows the convention of the other library dependencies.)

Issue: 2102
Submitted by: David James
Reviewed/Tweaked by:  Justin Erenkrantz
```

C2. Missing Directories

Patch committed in r11047 by Greg Hudson [22]

```
Fix issue #2032 (javahl build can fail due to missing directories) by
always calling mkdir-init at configure time.  Also fix a problem where
bindings targets would always rebuild because they depended on mkdir-init.
From David James <james@cs.toronto.edu>.

* configure.in: Always call mkdir-init, even if we are not in a VPATH
  setup.  This fixes issue 2032.
* build.conf: Removed mkdir-init from all dependency lists because
  it's now always run inside configure.  This change also prevents make
  from unnecessarily recompiling targets just because the file
  'mkdir-init' does not exist.
```

C3. Test Suite Fails Due to Typo

Patch committed in r10773 by Patrick Mayweg [50].

```
* subversion/bindings/java/javahl/src/org/tigris/subversion/javahl/tests/
    BasicTests.java
   testBasicLogMessage : fix typo in expected log message
Patch thanks to David James.
```

C4. Can't Test Before Install

I submitted the following patch to the Subversion development list on November 25 [37]:

```
This patch solves issue #2040 by fixing JavaHL so that it can be
tested before it is installed. I've revised this patch in response to
feedback from Justin Erenkrantz and I hope that this version is much
improved.
* Makefile.in
 (check-javahl): Instead of using the installed version of JavaHL, use
    the version compiled by libtool in the @JAVAHL_OBJDIR@ directory.
    We use FIX_JAVAHL_LIB to link libsvnjavahl-1.jnilib to
    libsvnjavahl-1.dylib on Mac OS X. We also replace the hardcoded
    path to the javahl classes directory with $(javahl_tests_PATH).
* configure.in
 (JAVAHL_OBJDIR): Get the directory where the compiled version of the
    JavaHL library is stored and save it as @JAVAHL_OBJDIR@
 (FIX_JAVAHL_LIB): On Mac OS X, substitute @FIX_JAVAHL_LIB@ for code
    to link libsvnjavahl-1.jnilib to libsvnjavahl-1.dylib
* build/generator/gen_make.py
 (Generator.write): Output the name of the directory where each target
    generates its files as %s_PATH. E.g., for the libsvnjavahl target,
    we create a variable called libsvnjavahl_PATH. Also, by setting
    path = target_ob.output_dir for TargetJava objects, we ensure that
    targets of type TargetJavaHeaders output the correct output
    directory into the %s_PATH variable.

Discussion:
- For why we need to link libsvnjavahl-1.jnilib to libsvnjavahl-1.dylib
  on Mac OS X, see Issue 1632:
```

```
  <http://subversion.tigris.org/issues/show_bug.cgi?id=1632>
- We assume that libtool stores its libraries in
    '$(libsvnjavahl_PATH)'/$objdir
  According to the libtool documentation, the $objdir variable
  specifies the name of the directory where the compiled versions
  of the library are stored. I also checked the libtool source code and
  found that regardless of your platform, when you try to run an
  uninstalled program using libtool, it simply adds the location of the
  $objdir directory to the variable specified by $shlibpath_var.
  Therefore there is no need for complex code to extract this directory
  by running libtool and checking $shlibpath_var; all we need to do is
  look in $objdir. So that's what we do.
- For discussion regarding the best method for retrieving the $objdir
  variable from libtool, see
  <http://svn.haxx.se/dev/archive-2004-11/1074.shtml>
- If my change "path = target_ob.output_dir" is controversial, feel
  free to omit it. It is only a cosmetic fix to the output of %s_PATH.
- This patch has been substantially improved from previous versions
  thanks to feedback from Justin Erenkrantz, Ben Reser, and Greg
  Hudson. For full history of the development of this patch see issue
  #2040: <http://subversion.tigris.org/issues/show_bug.cgi?id=2040>
- Let me know if you have any feedback. I am always happy to revise my
  patches to meet your needs.
```

C5. Slow Compilation

Patch committed in r12018 by Justin Erenkrantz [51]

```
Resolve Issue #2103 by teaching the javahl build section about dependencies.
* build.conf (javahl-javah): Pass -force to javah to always update .h file.
  Otherwise, if there were no changes, then javah wouldn't touch the file and
  make would not recognize that the file is in sync.
* build/generator/gen_base.py
  (gen_base.TargetJavaClasses.add_dependencies): Add class file to dependency
   list instead of the source file.
* build/generator/gen_make.py
  (gen_make.Generator.write): Switched TargetJavaClasses rules to
     depend on the object files instead of the source files. These
     object files are generated in a single call to 'javac'. This
     change prevents unnecessary regeneration of .class files.
  (gen_make.Generator.write): Switched TargetJavaHeaders rules to
     depend on the header files instead of the source files. These
     header files are generated in a single call to 'javah'. This
     change prevents unnecessary regeneration of .h files.
  (gen_make.Generator.write): Added code to keep track of
     dependencies for new TargetJavaHeaders and TargetJavaClasses rules
(Justin added the build.conf tweak in order to make it really work and
 tweaked some comments in gen_base.py and gen_make.py to make it more
 understandable.)
Issue: 2103
Submitted by: David James
Reviewed by: Justin Erenkrantz
```

Appendix D: References

[1] Eaton, Dave. *Configuration Management Frequently Asked Questions*.
URL: ftp://ftp.cs.uu.nl/pub/NEWS.ANSWERS/sw-config-mgmt/faq

[2] *Perforce 98.2: Glossary*. Perforce.
URL: http://www.perforce.com/perforce/doc.982/cmdguide/glossary.html

[3] *Helium Project Overview*. Third Bit.
URL: http://www.third-bit.com/helium/doc/overview.html

[4] Coyner, Brian M. (July 14, 2004). *Building Subversion With Java Bindings On Mac OS X*. O'Reilly Developer Weblogs. URL: http://www.oreillynet.com/pub/wlg/5210

[5] James, David. (2004-12-12) *[RFC] If bindings are enabled, install them with standard 'install' target*
URL: http://svn.haxx.se/dev/archive-2004-12/0504.shtml

[6] James, David. (2004-10-31) *Request for 1.1.2 nomination: Simple but important JavaHL installation fixes in r11047, r11050*. URL: http://svn.haxx.se/dev/archive-2004-10/1597.shtml

[7] Hudson, Greg. (2004-12-04) *Re: Why do we allowed mixed versions of libsvn_* ?*
URL: http://svn.haxx.se/dev/archive-2004-12/0184.shtml

[8] Kopp, David. (2004-08-31) *Minor javahl build breakage.*
URL: http://subversion.tigris.org/issues/show_bug.cgi?id=2032

[9] James, David. (2004-08-29) *[PATCH] javaHL testBasicLogMessage test fails.*
URL: http://svn.haxx.se/dev/archive-2004-08/0821.shtml

[10] James, David. (2004-09-02) *Issue 2040: Can't Test JavaHL Bindings Before Installation.*
URL: http://subversion.tigris.org/issues/show_bug.cgi?id=2040

[11] James, David. (2004-10-17) *Prevent unnecessary regeneration of JavaHL .class and .h files.*
URL: http://subversion.tigris.org/issues/show_bug.cgi?id=2103

[12] James, David. (2004-09-02) *[SVN RFC] Making JavaHL easier to maintain?*
URL: http://svn.haxx.se/dev/archive-2004-09/0054.shtml

[13] Erenkrantz, Justin. (2004-09-02) *Re: [SVN RFC] Making JavaHL easier to maintain?*
URL: http://svn.haxx.se/dev/archive-2004-09/0058.shtml

[14] Bowsher, Max. (2004-09-02) *Re: [SVN RFC] Making JavaHL easier to maintain?*
URL: http://svn.haxx.se/dev/archive-2004-09/0070.shtml

[15] James, David. (2004-09-02) *[RFC] Streamlining Binding Installation.*
URL: http://svn.haxx.se/dev/archive-2004-09/0075.shtml

[16] Huelsmann, Erik. (2004-09-02) *Re: [RFC] Streamlining Binding Installation*.
URL: http://svn.haxx.se/dev/archive-2004-09/0079.shtml

[17] Reser, Ben. (2004-09-03) *Re: [RFC] Streamlining Binding Installation*.
URL: http://svn.haxx.se/dev/archive-2004-09/0111.shtml

[18] James, David. (2004-09-19) *[PATCH] Allow JavaHL to be installed separately from main Subversion package (was: Re: [PATCH] Fix cross-library installation dependencies for libsvn_fs and libsvn_fs_base)*
URL: http://svn.haxx.se/dev/archive-2004-09/0656.shtml

[19] Hudson, Greg. (2004-09-19) *Add install dependencies for libsvn_fs and libsvn_fs_base.*
URL: http://svn.collab.net/viewcvs/svn?rev=11050&view=rev

[20] Thon, Holger. (2004-09-13) *[PATCH] javahl fixes (Issue #2040) (Issue #2032).*
URL: http://svn.haxx.se/dev/archive-2004-09/0408.shtml

[21] James, David. (2004-09-18) *[PATCH] Always call mkdir-init, even if we're not in a VPATH setup (Fixes Issue #2032).* URL: http://svn.haxx.se/dev/archive-2004-09/0633.shtml

[22] Hudson, Greg. (2004-09-19) *Fix issue #2032 by always calling mkdir-init at configure time*.
URL: http://svn.collab.net/viewcvs/svn?rev=11047&view=rev

[23] Bowsher, Max. (2004-09-22) Merge r10773 from trunk to 1.1.x branch.
URL: http://svn.collab.net/viewcvs/svn/?rev=11091&view=rev

[24] Pilato, C. (2004-09-02) *Re: [RFC] Streamlining Binding Installation.*
URL: http://svn.haxx.se/dev/archive-2004-09/0084.shtml

[25] Erenkrantz, Justin. (2004-09-02) *Re: [RFC] Streamlining Binding Installation.*
URL: http://svn.haxx.se/dev/archive-2004-09/0088.shtml

[26] Pilato, C. (2004-09-02) *Re: [RFC] Streamlining Binding Installation.*
URL: http://svn.haxx.se/dev/archive-2004-09/0093.shtml

[27] Reser, Ben. (2004-09-13) *Re: [PATCH] Can't test JavaHL bindings before installation (Issue #2040)* URL: http://svn.haxx.se/dev/archive-2004-09/0401.shtml

[28] Thon, Holger. (2004-09-12) *Re: [PATCH] Can't test JavaHL bindings before installation (Issue #2040)* URL: http://svn.haxx.se/dev/archive-2004-09/0398.shtml

[29] Hudson, Greg. (2004-09-12) *Re: [PATCH] Can't test JavaHL bindings before installation (Issue #2040)* URL: http://svn.haxx.se/dev/archive-2004-09/0391.shtml

[30] Reser, Ben. (2004-09-13) *Re: [PATCH] Can't test JavaHL bindings before installation (Issue #2040)* URL: http://svn.haxx.se/dev/archive-2004-09/0402.shtml

[31] Thon, Holger. (2004-09-13) *[PATCH] javahl fixes (Issue #2040) (Issue #2032).* URL: http://svn.haxx.se/dev/archive-2004-09/0408.shtml

[32] James, David. (2004-09-19) *[PATCH] Allow users to test JavaHL bindings before installation.* URL: http://svn.haxx.se/dev/archive-2004-09/0654.shtml

[33] Fogel, Karl. (2004-11-22) *JavaHL bindings status.* URL: http://svn.haxx.se/dev/archive-2004-11/0955.shtml

[34] Erenkrantz, Justin. (2004-11-24) *Re: JavaHL bindings status.* URL: http://svn.haxx.se/dev/archive-2004-11/1013.shtml

[35] *libtool script contents.* URL: http://www.delorie.com/gnu/docs/libtool/libtool_71.html

[36] James, David. (2004-11-25) *Best way to get "objdir" variable from libtool?* URL: http://svn.haxx.se/dev/archive-2004-11/1074.shtml

[37] James, David. (2004-11-25) *[PATCH] Allow JavaHL to be tested before installation (Solves Issue #2040)* URL: http://svn.haxx.se/dev/archive-2004-11/1075.shtml

[38] James, David. (2004-09-02) *Issue 2039: Fix JavaHL Dependencies* URL: http://subversion.tigris.org/issues/show_bug.cgi?id=2039

[39] Erenkrantz, Justin. (2004-11-24) *Resolve Issue #2103 by teaching the javahl build system about dependencies.* URL: http://svn.collab.net/viewcvs/svn?rev=12018&view=rev

[40] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion.* URL: http://svnbook.red-bean.com

[41] *Creative Commons Deed*. URL: http://creativecommons.org/licenses/by/2.0/

[42] James, David. (2004-09-18). *[PATCH] Autobuild dependencies before install.* URL: http://svn.haxx.se/dev/archive-2004-09/0609.shtml

[43] *Apache Ant – Welcome.* Apache Foundation. URL: http://ant.apache.org/

[44] Quinton, Reg. *How to write a Makefile.* URL: http://vertigo.hsrl.rutgers.edu/ug/make_help.html

[45] *Hippo: Neon.* URL: http://pyre.third-bit.com/hippo

[46] *GNU Libtool*. Free Software Foundation (FSF).

http://www.gnu.org/software/libtool/libtool.html

[47] *Subversion*. Apache Foundation. http://subversion.tigris.org/

[48] *SVNClientAdapter.* Subclipse: A Subversion Eclipse Plugin.

URL: http://subclipse.tigris.org/svnClientAdapter.html

[49] Erenkrantz, Justin. *Resolve Issue #2102 by adding the right dependencies for javahl bindings.*

URL: http://svn.collab.net/viewcvs/svn?rev=12016&view=rev

[50] Mayweg, Patrick. *Fix typo in expected log message.*

URL: http://svn.collab.net/viewcvs/svn?rev=10773&view=rev

[51] Erenkrantz, Justin. *Resolve Issue #2103 by teaching the javahl build section about dependencies.*

URL: http://svn.collab.net/viewcvs/svn?rev=12016&view=rev