Versionsverwaltung mit Subversion

Florian Wörter*

30. Juli 2007

*http://www.woerter.at

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Versionsverwaltung . . . 3
1.1.1 Allgemeines . . . 3
1.1.2 Geschichtliches . . . 4
1.2 Subversion . . . 5

2 Subversion - Die theoretische Sicht **6**
2.1 Was ist Subversion? . . . 6
2.1.1 Features von Subversion . . . 6
2.1.2 Die Architektur von Subversion . . . 8
2.2 Grundlegende Konzepte . . . 9
2.2.1 Das Repository . . . 9
2.2.2 Copy-Modify-Merge Modell . . . 9

3 Subversion - Die praktische Sicht **13**
3.1 Installation . . . 13
3.2 Komponenten . . . 13
3.3 Anlegen eines Subversion Repositories . . . 14
3.4 Zugriff auf das Repository . . . 15
3.5 Typische Vorgehensweise (Kommandozeile) . . . 15
3.5.1 Erzeugen eines Repositories und Importieren von Dateien . . . 15
3.5.2 Ein Backup des Repositories anlegen . . . 15
3.5.3 Wiederherstellen eines Backups . . . 16
3.5.4 Ein Checkout durchführen . . . 16
3.5.5 Aktualisierung der Working Copy . . . 16
3.5.6 Rückgängigmachen der Änderungen . . . 16
3.5.7 Speichern der Änderungen im Repository . . . 17
3.6 Typische Vorgehensweise (Eclipse) . . . 17

Literaturverzeichnis **19**

Kapitel 1

Einleitung

1.1 Versionsverwaltung

1.1.1 Allgemeines

Heute ist es in der Regel üblich, dass mehrere Entwickler an einem Softwareprojekt beteiligt sind. Dabei kommt es oft vor, dass mehrere Leute gleichzeitig an einer Quelltext-Datei arbeiten wollen / müssen, wobei es immer wieder zu den selben Problemen kommt, egal ob die Programmierer nun im gleichen Gebäude sitzen, oder sie tausende von Kilometer weit auf der Erde verstreut sind. Während ein Entwickler an einer Quelldatei arbeitet, kann es sehr leicht passieren, dass ein anderer Entwickler auch etwas an der Quelldatei ändert. Kopiert nun der Entwickler, der später fertig wird seine geänderte Quellcode-Datei wieder auf den Server, so überschreibt er somit die Änderungen des anderen Entwicklers. Eine Veranschaulichung dieses Problems ist in Abbildung 2.3 auf Seite 10 zu sehen. Dieses Problem tritt vorallem bei der Open Source Gemeinde verstärkt auf, da es hier üblich ist, dass die Entwickler auf der ganzen Erde verstreut sind und somit oft gar nicht wissen, wer genau jetzt noch aller an diesem Projekt arbeitet. Deshalb kommt auch die Lösung für dieses Problem aus der Open Source Gemeinde.
Das Zauberwort heisst *Versionsverwaltung*. Dabei handelt es sich um Systeme, die den Sourcecode zentral auf einen Server lagern. Will ein Entwickler an einer Datei arbeiten, so *checkt er die Datei aus* und lädt sie so auf seinen lokalem Rechner. Die Dateien werden mit *Versionsnummern* bezeichnet, so weiss der Entwickler sofort, welche Version dieser Datei er gerade bearbeitet. Hat der Programmierer seine Änderungen abgeschlossen, so führt er ein *Commit* durch. Dadurch wird die Datei wieder auf den Server, in das sogenannte *Repository* gespeichert. Bevor dies geschieht, wird jedoch vom System überprüft, ob die Datei inzwischen von jemand anderen bearbeitet wurde. Dies geschieht auf Basis der Versionsnummer, die für jedes im Repository befindliches Objekt vergeben und gespeichert wird. Stimmt die Versionsnummer überein, so wird diese inkrementiert und die geänderte Datei wird im Repository gespeichert. Ist jedoch schon eine neuere

Version der Datei im Repository, so muss ein sogenanntes *diff* durchgeführt werden. Dabei werden dem Entwickler die neue und die alte Version der Datei gezeigt und die Differenzen der beiden Dateien werden hervorgehoben. Nun muss der Entwickler entscheiden, welche der von ihm getätigten Änderungen übernommen werden sollen und welche nicht. Dabei muss er darauf achten, dass nur seine Änderungen übernommen werden.
Das Verwenden eines solchen Tools bringt jedoch noch eine Reihe von anderen nicht zu unterschätzenden Vorteile mit sich. Da die verschiedenen Versionen der Dateien auf dem Server gespeichert bleiben, ist es jederzeit möglich eine bestimmte Version des Projektes wiederherzustellen. Dadurch wird die Verwendung einer Versionsverwaltung sogar für eine Einzelperson schon interessant.
Um Platz zu sparen, speichert eine intelligente Versionsverwaltung (wie zum Beispiel Subversion) nur die Unterschiede zwischen einzelnen Dateiversionen.

1.1.2 Geschichtliches

SCCS

Das *Source Code Control System* wurde 1975 von AT&T entwickelt, und ist somit das erste Open Source Versionsverwaltungssystem. Es beinhaltete schon viele Mechanismen, die auch heute noch zum Einsatz kommen (Versionsnummern, Branching, Diffing).

RCS

Das *Revision Control System* wurde 1985 von *Walter Tichy* an der Purdue University entwickelt und ist ebenfalls ein Open Source Versionsverwaltungssystem. Dieses System baut auf das 10 Jahre früher entwickelte SCCS auf und erweitert seine Funktionalitäten. Erstmals ist es möglich, dass mehrere Personen gleichzeitig Kopien der Quellcode-Dateien besitzen können, auch wenn es nur eine schreibbare Version geben durfte.

CVS

Das *Current Version System* wurde 1986 von den Herren *Grune*, *Berlin* und *Polk* entwickelt. Dieses System war sehr lange das am weitesten verbreiteste Open Source Versionsverwaltungssystem. Es baut auf RCS auf und erweitert seine Funktionalitäten. Erstmals ist es möglich, dass mehrere Entwickler gleichzeitig an gleichen Dateien arbeiten und diese verändern. Auch neu war das Repository-Konzept. Dabei handelt es

sich um ein Quellverzeichnis, worauf Entwickler über das Netzwerk zugreifen konnten. Mein Kollege, Christoph Spielmann, wird in seiner Seminararbeit über CVS näher auf dieses System eingehen.

Kommerzielle Versionsverwaltungssysteme

Neben den oben beschriebenen Open Source Versionsverwaltungssystemen gibt es natürlich auch eine Reihe von kommerziellen Produkten. Einige Beispiele wären [2]:

- Microsoft Visual Source Safe
- Visible Software Razor
- Rational Clear Case
- Telelogic Continuus
- MKS Source Integrity
- Merant PVCS
- Perforce

1.2 Subversion

Subversion ist neben CVS das wohl bekannteste Open Source Versionsverwaltungssystem. Laut der Herstellerhomepage [1] will das System ein besseres CVS System sein. Dabei macht es sehr viel gleich wie CVS und versucht die grössten Mankos von CVS intelligenter zu lösen.
Ich werde in den folgenden Seiten versuchen, Ihnen einen kleinen Einblick in *Subversion* zu geben, und seine Vor- und Nachteile etwas näher zu erläutern.

Kapitel 2

Subversion - Die theoretische Sicht

2.1 Was ist Subversion?

Laut der Herstellerhomepage [1] soll Subversion ein besseres CVS System sein. Daher werden viele Probleme ähnlich wie bei CVS gelöst. Subversion (kurz *SVN* versucht jedoch bei Dingen, wo CVS Probleme hat, konsequent andere, einfachere Wege zu gehen. Leute, die bis jetzt mit CVS gearbeitet haben, werden keine grossen Probleme beim Umstieg zu Subversion haben. Laut [2] ist es sogar möglich, CVS Repositories nach Subversion zu konvertieren.
Subversion wird seit 2000 von CollabNet, Inc. entwickelt, liegt in einer Apache/BSD ähnlichen Open Source Lizenz vor und ist derzeit in der Version 1.3.2 zum Download verfügbar.
Mit Subversion kann jede Art von Daten (auch Binärdateien) versioniert werden. Es ist nicht auf Quellcode beschränkt.

2.1.1 Features von Subversion

Hier sind vorallem Features ausgewählt, bei denen sich Subversion von CVS unterscheidet. Die Features wurden sinngemäß aus *Version Control with Subversion* [3] übernommen.

Directory versioning

Im Gegensatz zu CVS kann Subversion nicht nur einzelne Dateien, sondern auch auch Verzeichnisse versionisieren. Veränderungen an ganzen Verzeichnisbäumen werden erkannt und verarbeitet.

True version history

Da bei CVS nur einzelne Dateien versionisiert werden, können Veränderungen wie Kopieren, Verschieben oder umbenennen von Dateien nicht verfolgt werden – bei Subversion schon. Zusätzlich beginnt bei Subversion für jede neu erstellte Datei eine völlig neue History. Bei CVS war es nicht möglich, eine neue Datei anzulegen mit einen Namen, den früher einmal eine Datei hatte, ohne die History dieser (vielleicht völlig unrelevanten Datei) mitschleppen zu müssen.

Atomic commits

Commits sind bei Subversion atomar. Das heisst, entweder werden alle Veränderungen eines Commits ins Repository übernommen, oder gar keine. Deses Feature hilft viele Probleme zu vermeiden, die durch nicht vollständig durchgeführte Commits entstehen können.

Versioned metadata

Jedes Objekt im Repository hat bestimmte Eigenschaften. Eigenschaften werden immer als Schlüssel-Wert Tuppel abgespeichert. Man kann so viele solche Tuppel wie man will zu einem Objekt abspeichern.

Choice of network layers

Der Zugriff auf das Subversion Repository ist sehr modular und abstrakt aufgebaut. Dadurch können Leute recht einfach eigene netzwerkbasierte Zugriffsarten auf das Repository implementieren. Zum Beispiel kann Subversion als ein *Apache Extension Module* in Zusammenarbeit mit dem Apache Webserver verwendet werden.

Consistent data handling

Der diff-Algorithmus von Subversion arbeitet sowohl für Klartext- wie auch für Binärdateien. Es werden bei beiden Arten von Dateien nur die Änderungen auf dem Server gespeichert und über das Netzwerk übertragen.

Efficient branching and tagging

Beim Erstellen von Branches oder Tags verwendet Subversion eine den Hardlinks ähnliche Technik. Dadurch wird für das Erstellen von Branches oder Tags nur sehr wenig Zeit benötigt.

2.1.2 Die Architektur von Subversion

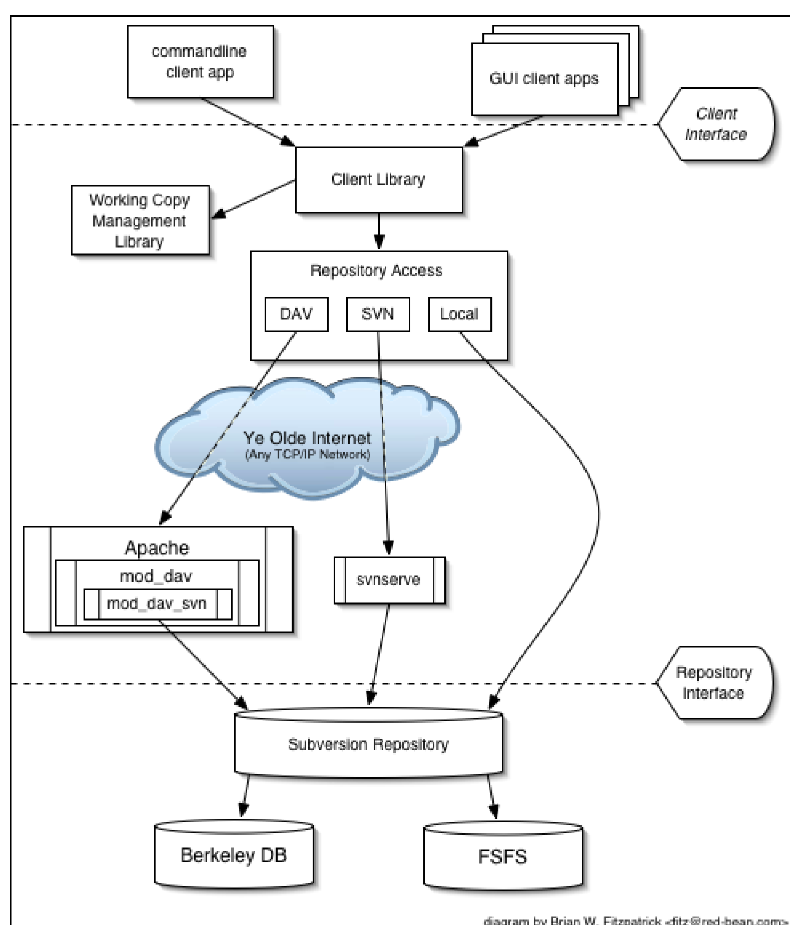

Abbildung 2.1: Subversion Architektur [3]

In der Abbildung 2.1 sehen wir die grundlegende Architektur von Subversion. Am unteren Ende dieser Abbildung sehen wir das Repository. Ganz oben befindet sich der Client. Die Daten, die der Client gerade bearbeitet, werden auch *Working Copies* genannt. Dazwischen befinden sich die sogenannten *Repository Access Layers*. Jenachdem, wie jemand auf das Repository zugreift, kann er verschiedene Layer durchlaufen.

2.2 Grundlegende Konzepte

2.2.1 Das Repository

Wie schon erwähnt, bildet das Repository den grundlegenden Kern von Subversion. Alle Informationen über die einzelnen Dateien und Verzeichnisse eines Projektes werden im dazugehörigen Repository abgespeichert. Dieses wird zentral verwaltet und speichert die Informationen hirarchisch ab. Man könnte sagen, das Repository speichert eine Reihe von Verzeichnisbäumen. Jeder Verzeichnisbaum hat dabei eine eindeutige Revisionsnummer und stellt sozusagen eine Momentaufnahme des Repositories dar. In der Abbildung 2.2 sehen wir, wie das Konzept mit den Revisionsnummern funktioniert. Die Zahlen oben in der Grafik stellen dabei die aktuelle Revisionsnummer, der Baum darunter das aktuelle Repository dar. Abgespeichert werden die Daten meistens in einer Berkeley DB4 Datenbank [2].

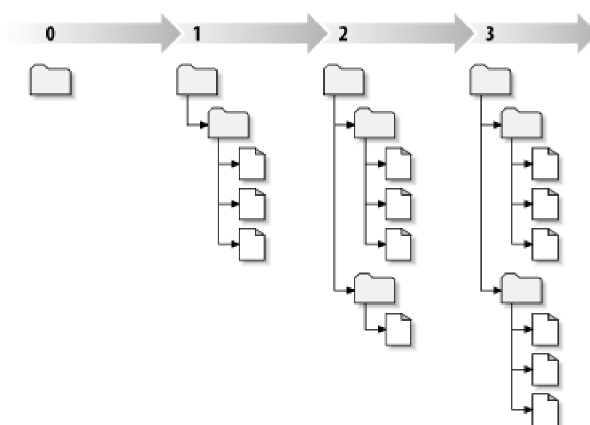

Abbildung 2.2: Subversion Repository [2]

2.2.2 Copy-Modify-Merge Modell

Alle Versionsverwaltungssysteme haben ein gemeinsames Problem, das es zu lösen gilt: *Wie kann ich den Benutzern alle Daten zur Verfügung stellen, ohne dass sie sich gegenseitig auf die Füße treten?* In Abbildung 2.3 auf der nächsten Seite wird dieses Problem veranschaulicht. Eine nähere Beschreibung dieses Problemes finden Sie unter 1.1.1 auf Seite 3 („Allgemeines“). Kurz gesagt würde Sally die Änderungen von Harry überschreiben.

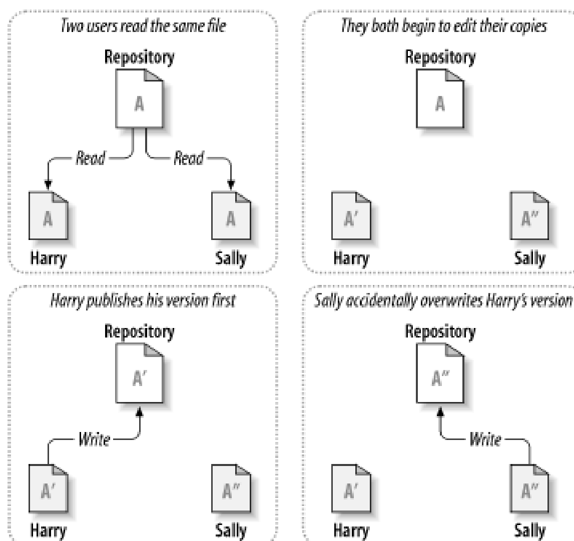

Abbildung 2.3: Das Filesharing Problem [3]

Um dieses Problem zu lösen, gibt es mehrere Ansätze. Viele Versionsverwaltungssysteme verwenden zur Lösung des Filesharing Problems ein sogenanntes *Lock-Modify-Unlock* Modell. Dieses Modell ist in Abbildung 2.4 veranschaulicht.

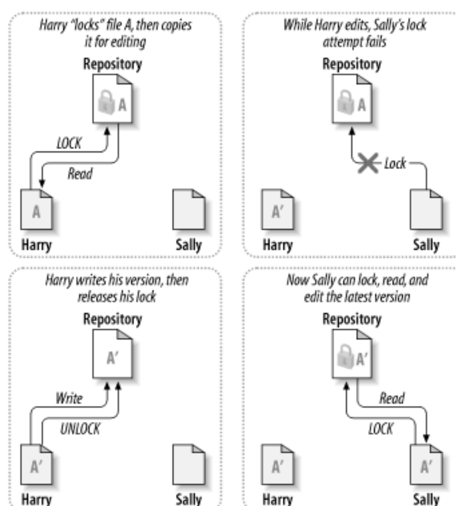

Abbildung 2.4: Das Lock-Modify-Unlock Modell [3]

Dabei ist das gleichzeitige Bearbeiten von Dateien praktisch nicht möglich. Benutzer Harry *sperrt* das Dokument und bearbeitet es. Erst wenn er es fertig bearbeitet und somit *entsperrt* hat, kann Sally es sperren und verändern. Sally kann das Dokument zwar lesen, während es von Harry bearbeitet wird, jedoch nicht verändern. Es liegt auf der Hand, das dieses Modell einige Nachteile mit sich bringt.

Subversion verwendet, wie übrigens CVS auch, das sogenannte *Copy-Modify-Merge Modell*. Dabei hält jeder Benutzer eine lokale Kopie des Repositories gespeichert. Die Benutzer arbeiten parallel lokal an ihren Kopien der Repositories. Letztendlich führen die Benutzer ein *Commit* durch, und die lokale Version eines Benutzers wird mit der derzeitigen Version im Repository *gemerged*. D.h. es werden nur Teile übernommen, die der Benutzer auch geändert hat. Subversion hilft einem zwar beim Merging, jedoch ist letztendlich der Benutzer dafür verantwortlich, dass die richtige Version in das Repository gelangt, indem er nur Sachen überschreibt, die er auch selbst seit dem letzten Checkout verändert hat. Der ganze Vorgang wird in den Abbildungen 2.5 und 2.6 detailiert dargestellt.

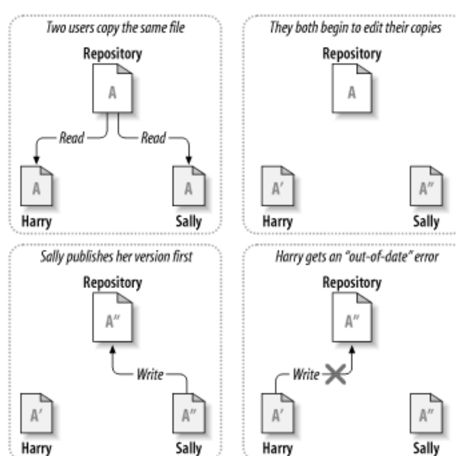

Abbildung 2.5: Das Copy-Modify-Merge Modell [3]

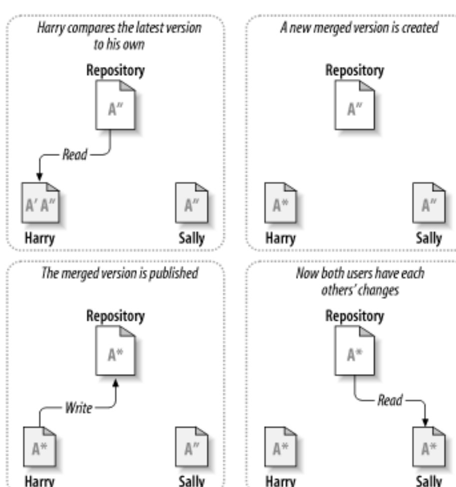

Abbildung 2.6: Das Copy-Modify-Merge Modell (Fortsetzung) [3]

Aber was passiert wenn sich die Änderungen von Harry mit denen von Sally überlappen? Diese Situation bezeichnet man als *Konflikt*. Die betreffende Datei wird gekennzeichnet und ist in beiden Versionen (der von Harry und der aktuellen Repository Version) für Harry lokal verfügbar. Harry kann nun beispielsweise Rücksprache mit Sally halten, das Problem lokal fixen und die Datei ins Repository geben. Es liegt auf der Hand, dass solche Konflikte nur von Menschen, nicht aber von Computerprogrammen, gelöst werden können.

Kapitel 3

Subversion - Die praktische Sicht

3.1 Installation

Subversion wurde auf Basis von APR (*Apache Protable Runtime library*) entwickelt, und ist somit auf sehr vielen Betriebssystemen lauffähig, auf denen auch ein Apache Server läuft. Unter http://subversion.tigris.org/project_packages.html findet man sowohl Binärdateien für Linux, IBM i5, Mac OS X und Windows, als auch den Sourcecode zum selber compilieren. Die unter obigen Link verfügbaren Dateien enthalten sowohl Server- als auch Clientkomponenten und Tools. Die Installation ist von Betriebssystem zu Betriebssystem unterschiedlich und es würde zu viel Zeit und Platz kosten an dieser Stelle näher darauf einzugehen. Im Sourcecode-Download ist jedoch im Toplevel eine INSTALL Datei, die für die jeweilige Plattform das Builden der Subversion Version beschreibt. Für Gentoo Benutzer befindet sich das Subversion Paket unter devutil/subversion und kann ganz bequem per emerge Befehl installiert werden.

3.2 Komponenten

Nach der erfolgreichen Installation stellt Subversion einige Komponenten zur Verfügung.

svn

Hierbei handelt es sich um ein Kommandozeilen Programm. Beim Arbeiten mit Subversion wird nur dieses Programm benötigt. Alle anderen Komponenten werden zur Administration von Subversion verwendet.

svnversion

Zeigt den Status der lokalen *Working Copy* an.

svnlook

Dient zur Anzeige des Inhaltes eines Subversion Repositories.

svnadmin

Dient zur Administration von Subversion.

svndumpfilter

Man hat mit dem Programm *svnadmin* die Möglichkeit sogenannte Dumpfiles zu erstellen. Diese kann man mit dem Programm *svndumpfilter* filtern und somit leichter durchsuchen.

mod_dav_svn

Dabei handelt es sich um ein Modul für den Apache Webserver. Man benötigt es, damit Benutzer über den HTTP Server auf das Repository zugreifen können.

svnserve

Dies ist ein eigenes Serverprogramm, das das Repository über das Netzwerk zugänglich macht.

Zusätzlich gibt es auf [1] noch ein Programm (*cvs2svn*), das CVS Repositories nach Subversion konvertieren kann.

3.3 Anlegen eines Subversion Repositories

Ein neues Repository kann mit dem Programm *svnadmin* angelegt werden.

```
1 $svnadmin create /usr/local/svn/repos
```

Dabei wird ein neues Repository im Verzeichnis /usr/local/svn/repos erzeugt. Die Daten werden mit obigen Befehl in einer Berkeley DB4 Datenbank abgelegt.

Mit dem zusätzlichen Parameter

```
--fs-type
```

kann man eine alternative Speicherungsart wählen.

3.4 Zugriff auf das Repository

Im folgenden Beispiel wird das vorhandene Verzeichnis namens meinprojekt in das Repository unter /usr/local/svn/repos/projekt1 importiert. Mit dem -m Parameter fügt man eine Notiz hinzu.

```
1 $svn import meinprojekt file:///usr/local/svn/repos/projekt1 -m "add meinprojekt"
2 Adding meinprojekt/test.c
3 Committed revision 1.
```

Natürlich kann man auch über andere Protokolle auf das Repository zugreifen . . .

```
1 $svn import meinprojekt http://subversionserver.de/usr/local/svn/repos/projekt1 -m "add meinprojekt"
2 Adding meinprojekt/test.c
3 Committed revision 1.
```

```
1 $svn import meinprojekt https://subversionserver.de/usr/local/svn/repos/projekt1 -m "add meinprojekt"
2 Adding meinprojekt/test.c
3 Committed revision 1.
```

```
1 $svn import meinprojekt svn://svnserver.de/usr/local/svn/repos/projekt1 -m "add meinprojekt"
2 Adding meinprojekt/test.c
3 Committed revision 1.
```

3.5 Typische Vorgehensweise (Kommandozeile)

3.5.1 Erzeugen eines Repositories und Importieren von Dateien

Wie man ein Repository erstellt und Dateien importiert haben wir in 3.3 auf der vorherigen Seite („Anlegen eines Subversion Repositories“) und 3.4 („Zugriff auf das Repository“) bereits geklärt.

3.5.2 Ein Backup des Repositories anlegen

Man kann sich ohne weiteres einen *Dump* des Repositories erstellen.

```
1 $svnadmin dump /usr/local/svn/repos/ > repos-dumpfile
2 * Dumped Revision 0.
3 * Dumped Revision 1.
```

Es ist ratsam, immer wieder einmal ein solches Backup seines Repositories durchzuführen.

3.5.3 Wiederherstellen eines Backups

Will man ein erstelltes Backup wiederherstellen, so macht man das mit dem folgenden Befehl:

```
1 $svnadmin load /usr/local/svn/ < repos-dumpfile
2 <<< Started new txn, based on original revision 1
3      * adding path : project1 ... done.
4      * adding path : project1/test.cpp ... done.
```

3.5.4 Ein Checkout durchführen

Bevor wir Dateien in einem Repository verändern können, müssen wir sie auschecken. Wir erstellen eine sogenannte *Working Copy*.

```
1 $svn checkout file:///usr/local/svn/ arbeitskopie
2 A arbeitskopie/project1
3 A arbeitskopie/project1/test.cpp
4 Committed revision 1.
```

Damit hat man eine lokale Kopie des Repositories im Ordner arbeitskopie.

3.5.5 Aktualisierung der Working Copy

Mit dem folgenden Befehl können wir die Working Copy aktualisieren.

```
1 $svn update /usr/local/arbeitskopie
2 Revision 1.
```

Nun kann man seine Änderungen an der lokalen Working Copy durchführen.

3.5.6 Rückgängigmachen der Änderungen

Ist man mit den Änderungen an der lokalen Working Copy nicht zufrieden, so kann man alle Änderungen auch wieder rückgängig machen.

```
1 $svn revert /usr/local/arbeitskopie
```

3.5.7 Speichern der Änderungen im Repository

```
1 $svn commit -m "added howto section."
2 Sending project1/test.cpp
3 Transmitting file data.
4 Committed revision 2.
```

Damit werden die Änderungen an der lokalen Arbeitskopie ins Repository übernommen. Man sieht, dass sich die Revision erhöht. Nach dem Commit sollte man nicht vergessen, seine lokale Working Copy wieder zu aktualisieren.

3.6 Typische Vorgehensweise (Eclipse)

Ich persönlich verwende für viele Programmierungsaufgaben (vorallem Java) die Eclipse Plattform [4]. Für diese IDE gibt es ein Plugin für das Verwenden von Subversion namens *Subclipse* [5].

Installieren kann man dieses Plugin über den in Eclipse integrierten Update Manager (im Help Menü von Eclipse). Dazu muss man http://subclipse.tigris.org/update_1.0.x als Update Site hinzufügen.

Ist Subclipse fertig installiert, so ist eine neue Perspective (*SVN Repository Exploring*) verfügbar, die man über Window-Open Perspective-Other aufrufen kann. In dieser Perspektive kann man sich nun mit Rechtsklick und New Repository Location zu einem Repository verbinden. Man muss nur die URL zum Repository angeben und schon erscheint das Repository in der SVN Repository Exploring Perspektive.

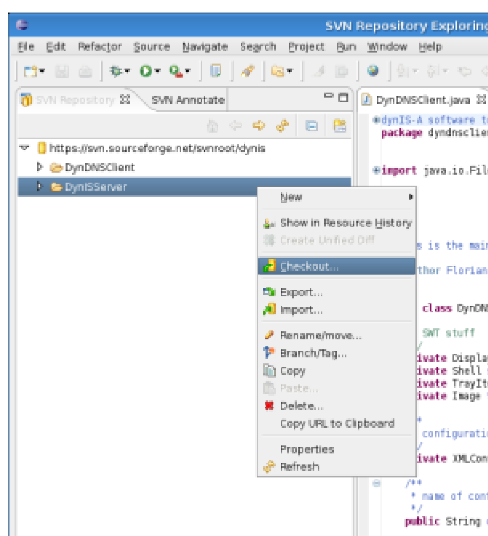

Abbildung 3.1: Die SVN Repository Exploring Perspective in Eclipse

Nun kann man mit Rechtsklick auf das entsprechende Projekt die Dateien auschecken. Zum Programmieren wechselt man am besten wieder in die Java Perspektive. Hat man seine Programmierarbeiten abgeschlossen, klickt man mit der rechten Maustaste auf das Projekt und wählte den Menüpunkte Team-Commit aus. Nun kann man optional noch eine Bemerkung angeben und die Änderungen werden (natürlich nach Überprüfung von Benutzername und Kennwort) ins Repository übernommen.

Literaturverzeichnis

[1] Tigris.org: Open Source Software Engineering Tools. http://subversion.tigris.org.

[2] Versionsverwaltung mit Subversion. Carla Schreiber, HTW Dresden, Allgemeine Informatik. http://wwwbs.informatik.htw-dresden.de/svortrag/i01/Schreiber/

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

[4] Die Eclipse IDE. http://www.eclipse.org.

[5] Das Subclipse Plugin für die Eclipse IDE. http://subclipse.tigris.org

Abbildungsverzeichnis

2.1 Subversion Architektur [3] 8
2.2 Subversion Repository [2] 9
2.3 Das Filesharing Problem [3] 10
2.4 Das Lock-Modify-Unlock Modell [3] 10
2.5 Das Copy-Modify-Merge Modell [3] 11
2.6 Das Copy-Modify-Merge Modell (Fortsetzung) [3] 11

3.1 Die SVN Repository Exploring Perspective in Eclipse 17