ganz kurze Einführung in CVS

Korbinian Bartels Klaus Jähne

16. Juli 2000

1 CVS — Kann man das essen?

CVS ist ein System zur Versionskontrolle, das für Softwareprojekte entwickelt wurde. Als ein solches sollte es bei folgenden Problemen als Werkzeug dienen:

- mehrere Benutzer arbeiten gleichzeitig an einem Projekt und bearbeiten die gleiche Datei, dabei tritt ein Konflikt auf.
- in einem Programm tritt ein Bug auf — seit wann gibt es den Bug, wie ist er entstanden?
- Entwickler möchten verteilt über das Internet ein Programm entwickeln.

Generell läßt sich also das folgende Anforderungsprofil für ein Versionsmanagementsystem definieren:

- Änderungen in Quelltexten sollen ersichtlich gemacht werden.
- alle Änderungen in Quelltexten sollen dokumentiert werden.
- alle älteren Versionen sollten wiederherstellbar sein.
- mehrere Benutzer sollen gleichzeitig an einem Projekt arbeiten können.

CVS steht im übrigen für *Concurrent Version System*.

2 Wie arbeitet CVS?

Ansatz CVS verwendet ein zentrales *Repository* (engl. für *Behälter*). Dieses befindet sich in einem Verzeichnis auf einem Rechner. Darin sind enthalten:

- für jedes Projekt ein Verzeichnis
- ein Verzeichnis für allgemeine Konfigurationsparameter (z.B. Benutzerverwaltung)

→ siehe Abb 1.

Client–Server CVS ist eine Client–Server–Anwendung. Das bedeutet:

- Das Repository wird von einem CVS–Server verwaltet.
- Die Benutzer verwenden einen CVS–Client, der wiederum über den CVS–Server auf das Repository zugreift.

→ siehe Abb 2.

Struktur des Repository

- das Repository enthält ein Verzeichnis für jedes Projekt
- im Projektverzeichnis wird die Verzeichnisstruktur des Projekts übernommen
- Jede Datei wird im jeweiligen Verzeichnis mit ihrem Namen abgespeichert — allerdings mit erweiterten Informationen:
  1. einer allgemeine Beschreibung der Datei
  2. Unterschieden zu vorherigen Versionen, damit sind alle älteren Versionen wiederherstellbar

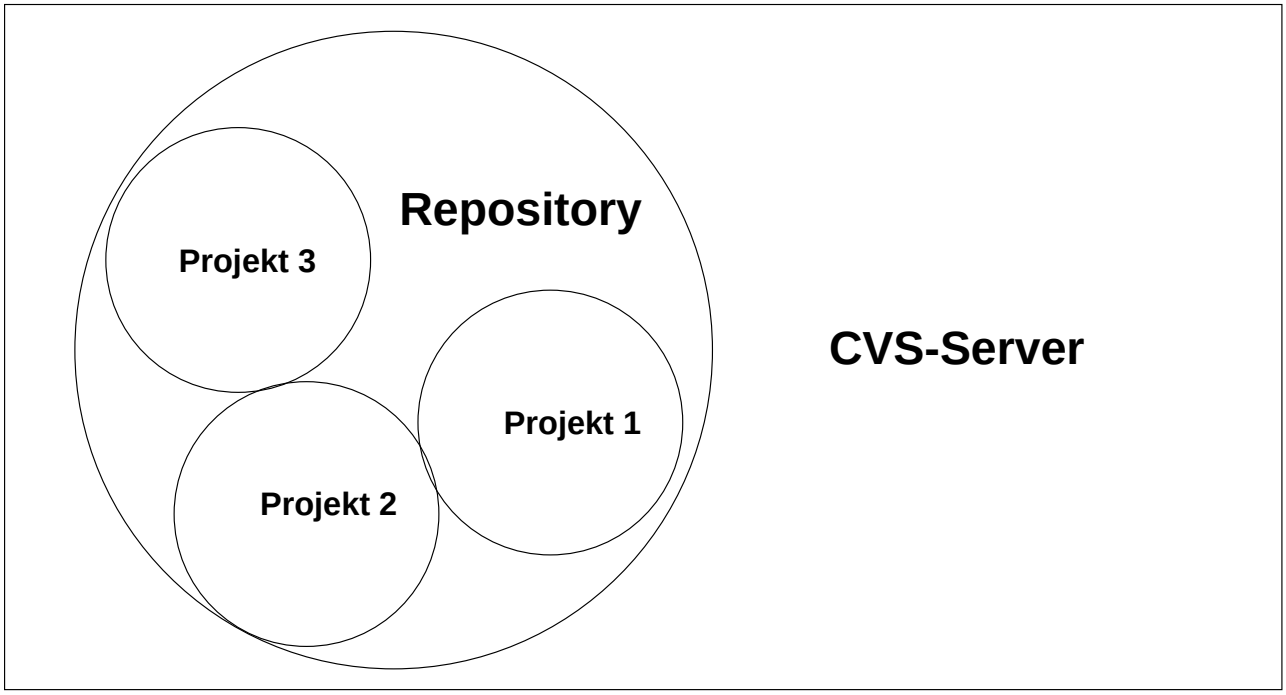

Abbildung 1: ein CVS–Repository

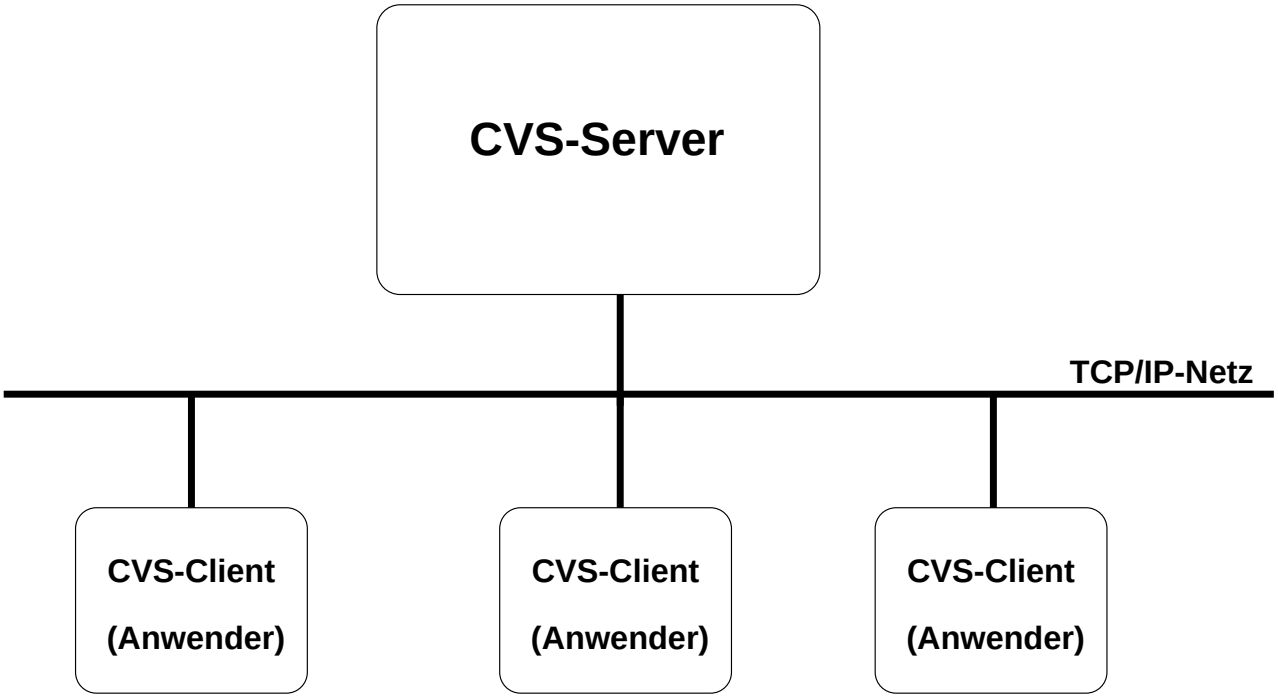

Abbildung 2: CVS als Client–Server–Applikation

3. dem Zeitpunkt jeder Änderung
4. Kommentare des Entwicklers zu jeder Änderung
5. dem Namen des ändernden Entwicklers
6. einer Versionsnummer

Zugriff auf CVS Um auf das CVS–Repository zuzugreifen, gibt es verschiedene Aktionen. Diese sollen im nächsten Teil vorgestellt werden.

3 CVS–Befehle

Die wichtigsten der ca. 50 CVS–Befehle lassen sich am einfachsten anhand einer CVS–Sitzung (siehe auch Abb. 3) zeigen:

login Als Benutzer anmelden:

```
$ export \
  CVSROOT=:pserver:kjaehne@cvs.pdv.fh-heilbronn.de:/usr/people/cvs/cvsroot/
$ cvs login
(Logging in to kjaehne@cvs.pdv.fh-heilbronn.de)
CVS password:
```

- Dem CVS–Client mitteilen, wo sich der Server befinet.
- Erst jetzt sind alle weiteren Aktionen möglich.

import Dateien unter CVS–Kontrolle stellen, neues Projekt anlegen

```
$ cd /home/kjaehne/to_import
$ cvs import -m "implementiert einen schonenden Röst-Algorithmus"\
  kroenung swp2 default
cvs server: Importing /usr/people/cvs/cvsroot/kroenung
N ./kroenung/coffe.java

No conflicts created by this import
```

Es wird übergeben:

- Eine Beschreibung des Projekts
- das zu importierende Verzeichnis
- ein *vendor tag*
- ein *release tag*

Alle Dateien aus dem Verzeichnis befinden sich jetzt im Repository.

checkout Dateien aus den Repository holen.

```
$ cd projects
$ cvs checkout kroenung
cvs server: Updating kroenung
U kroenung/coffe.java
```

- Die Dateien können jetzt bearbeitet werden.
- Über entsprechende Prameter können auch ältere Versionen ausgecheckt werden.

editieren, compilieren, debuggen etc. ...sind keine CVS–Befehle, sondern die nächsten Schritte, üblicherweise in der Entwicklungsumgebung

update Änderungen mit dem Repository abgleichen

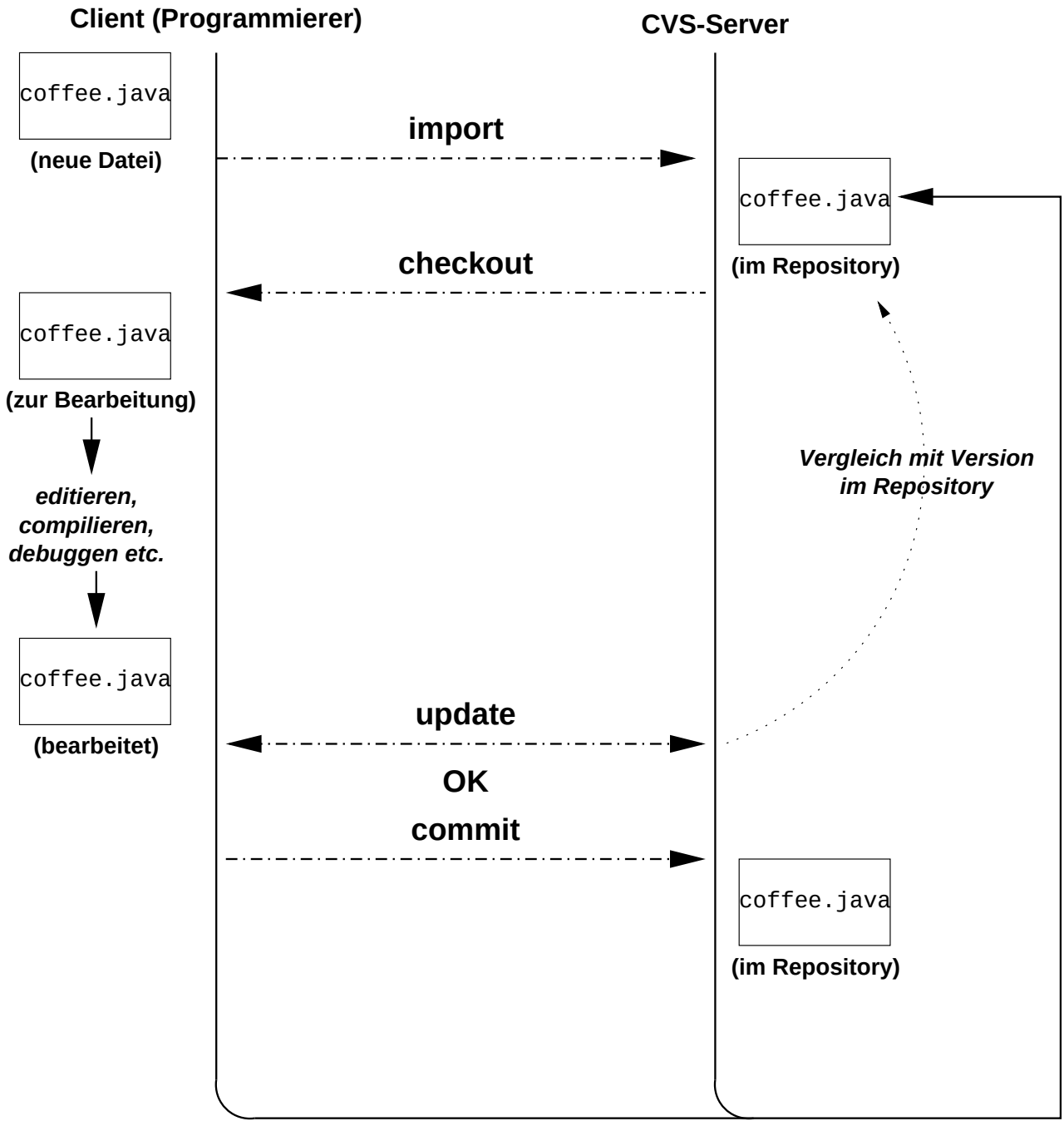

Abbildung 3: üblicher Verlauf einer CVS–Sitzung

```
$ cvs update
cvs server: Updating kroenung
M kroenung/coffe.java
```

- Dem CVS–Server werden alle Dateien im neuen Zustand bekannt gemacht
- falls Konfikte festgestellt werden, werden sie angezeigt. Der Entwickler muß sie dann von Hand beheben.
- falls es keine Konfikte gibt, kann man zum nächten Schritt übergehen

commit Änderungen ins Repository stellen.

```
$ cvs commit
cvs commit: Examining kroenung
Checking in kroenung/coffe.java;
/usr/people/cvs/cvsroot/kroenung/coffe.java,v  <--  coffe.java
new revision: 1.2; previous revision: 1.1
done
```

Dazwischen müssen in einem Editor die Änderungen protokolliert werden.

- Das Repository ist nun auf dem aktuellen Stand.
- Die Versionsnummer wurde erhöht.
- Andere Entwickler können nun mit dem neuen Code arbeiten.

log Veränderungen anzeigen

```
$ cvs log kroenung/coffe.java
RCS file: /usr/people/cvs/cvsroot/kroenung/coffe.java,v
Working file: kroenung/coffe.java
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        default: 1.1.1.1
        swp2: 1.1.1
keyword substitution: kv
total revisions: 3;     selected revisions: 3
description:
----------------------------
revision 1.2
date: 1999/12/06 23:27:08;  author: kjaehne;  state: Exp;  lines: +1 -1
- Bug in der Temperaturregelung behoben
----------------------------
revision 1.1
date: 1999/12/06 22:48:28;  author: kjaehne;  state: Exp;
branches:  1.1.1;
Initial revision
----------------------------
revision 1.1.1.1
date: 1999/12/06 22:48:28;  author: kjaehne;  state: Exp;  lines: +0 -0
Programm implemntiert einen schonenden Roest-Algorithmus
=============================================================================
```

- angezeigt werden alle Versionen, die es gab
- Änderungen:
  1. wann geändert wurde
  2. wer geändert hat

3. wieviel sich geändert hat
4. Kommentar des Entwicklers

- Metainformationen zur Datei

diff Unterschiede zwischen zwei Versionen anzeigen

```
$ cvs diff -c -r 1.1 -r 1.2 kroenung/coffe.java
Index: kroenung/coffe.java
===================================================================
RCS file: /usr/people/cvs/cvsroot/kroenung/coffe.java,v
retrieving revision 1.1
retrieving revision 1.2
diff -c -r1.1 -r1.2
*** coffe.java  1999/12/06 22:48:28     1.1
--- coffe.java  1999/12/06 23:27:08     1.2
***************
*** 24,30 ****
...
!     public static void main(String argv[])
...
--- 24,30 ----
...
!     public static final void main(String argv[])
...
```

Es wird angezeigt, was sich zwischen den Versionsständen geändert hat.

add neue Dateien ins Repository einbinden

```
$ cd kroenung
$ cvs add filter.java
cvs server: scheduling file ‘filter.java’ for addition
cvs server: use ’cvs commit’ to add this file permanently
$ cvs commit
cvs commit: Examining kroenung
RCS file: /usr/people/cvs/cvsroot/kroenung/filter.java,v
done
Checking in filter.java;
/usr/people/cvs/cvsroot/kroenung/filter.java,v  <--  filter.java
initial revision: 1.1
done
```

- Die Datei wird in das Repository eingebunden
- Mit dem *commit*–Befehl wird sie übernommen
- auch hier muß ein Kommentar eingegeben werden, der die Datei beschreibt

Konfilkte ...ist auch kein CVS–Befehl, hier zeigen wir aber, wir Konflikte vermieden werden. Deren Entstehung illustriert auch Abb. 4.

```
$ cvs update
cvs server: Updating kroenung
RCS file: /usr/people/cvs/cvsroot/kroenung/coffe.java,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into coffe.java
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in kroenung/coffe.java
C kroenung/coffe.java
$ vi kroenung/coffe.java
$ cvs commit
```

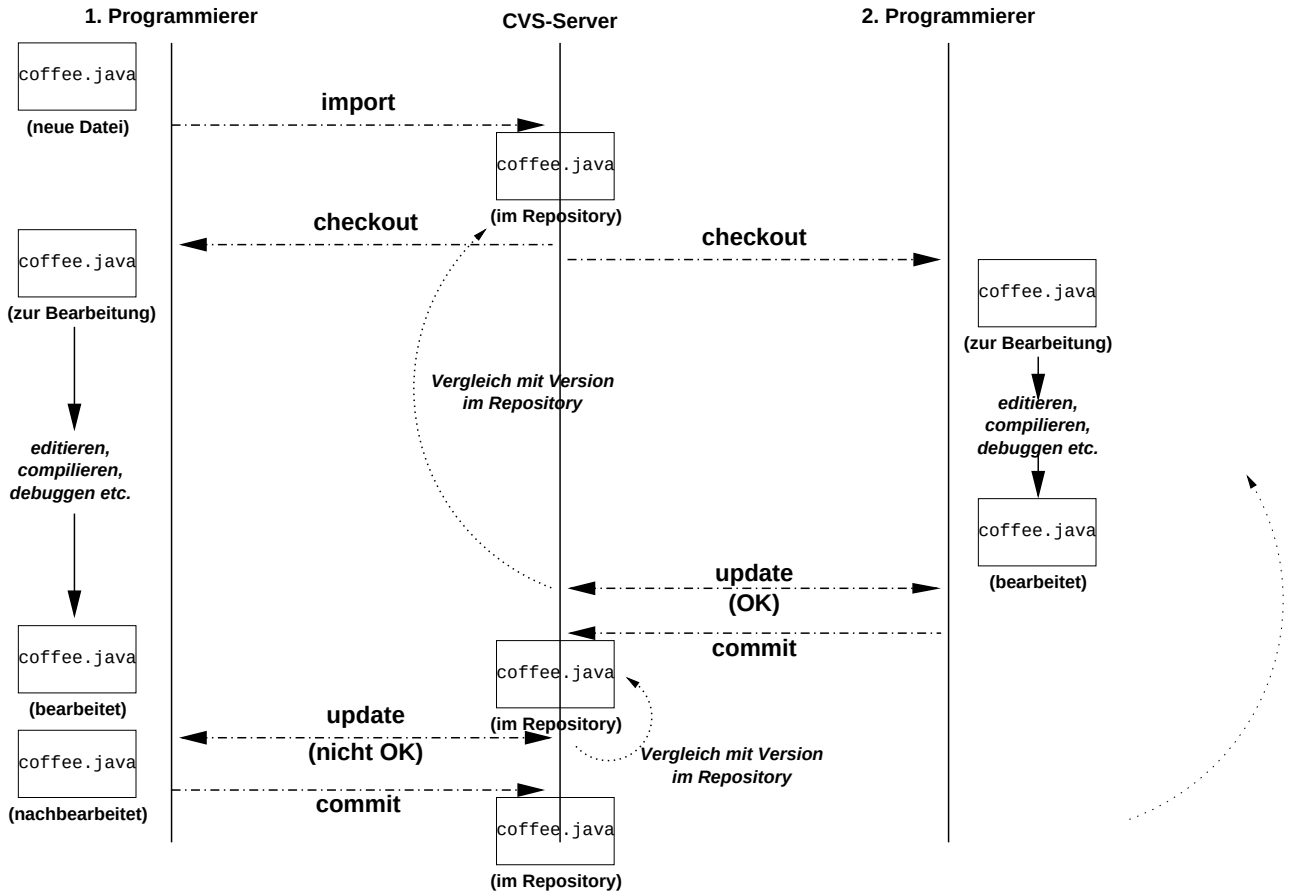

Abbildung 4: CVS–Sitzung mit Versionskonflikt

Es wurde also festgestellt, daß bereits andere Personen diese Datei an der gleichen Stelle verändert haben, und der Entwickler muß den Konflikt nun lösen.

Die Differenzen stehen nun im Sourcecode an der kritischen Stelle:

```
<<<<<<< coffe.java
    public static final int void main(String argv[])
=======
    void main(String argv[])
>>>>>>> 1.3
```

Wenn der Konflikt behoben ist, kann die Datei mit *commit* wieder eingecheckt werden.

Für mehr Komfort steht für Windows–Systeme *WinCVS* zur Verfügung, das in einem weiteren Teil im Detail vorgestellt wird.

Und hier noch die Adresse unseres CVS–Servers:

```
:pserver:<User-ID>@cvs.pdv.fh-heilbronn.de:/usr/people/cvs/cvsroot
```