Formal Modelling of Version Control Systems

By
David H. Kelk, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

Master of Computer Science
Department of Computing and Software
McMaster University

MASTER OF COMPUTER SCIENCE (2009) McMaster University
(Department of Computing and Software) Hamilton, Ontario

TITLE: Formal Modelling of Version Control Systems

AUTHOR: David H. Kelk, B.Sc. (York University)

SUPERVISOR: Dr. Emil Sekerinski

NUMBER OF PAGES: viii, 155

Abstract

Version control systems are widely used to manage collections of files and directories, along with changes made to them over their lifetime. Any previously checked in version of a file is recoverable at any time from the repository. They allow people to work on the same files in a decentralized and concurrent way, while consistently managing and integrating changes.

In this thesis we develop a subset of the SVN and CVS version control systems from specifications using Atelier B 4. Both of these systems are feature rich, widely used in cross-platform environments and representative of their class of file based extensional version control systems. Support for abstract data types like sets and refinement is well suited to the task. The most commonly used features such as Add, Check-in, Update are modeled in increasing detail in multiple refinement steps. Later refinement steps add features such as binary file support and the local cache. Having both models allows us to compare and contrast their feature sets.

Documentation for SVN and CVS is extensive but informal. One feature of CVS required experimentation when the written documentation was insufficient.

SVN is modelled in approximately 1400 lines in eight refinement steps with 109 proof obligations. CVS is likewise specified in roughly 1150 lines in seven steps with 29 proof obligations. With all proof obligations discharged we are confident the models represent the real systems and are a reasonable first step towards the goal of verifiable implementations of version control systems.

Acknowledgments

Dr. Emil Sekerinski, my supervisor, has been instrumental in seeing this thesis through to completion. His insights, explanations and feedback provide the small pushes allowing me to cross the big hurdles. A generous open door policy and good humor made the darkest days of fruitless proving brighter.

My former professors at York University were instrumental in my success at the graduate level: Dr. Ostroff introduced me to the combined awesomeness of Design by Contract and Eiffel. Dr. Tzerpos took a chance and agreed to mentor me for the independent study course for fourth year students. Dr. Wharton taught me to think in a programming language independent way with his two excellent courses in programming language fundamentals.

From my first undergraduate career I'm especially thankful for Professor Paul Delaney for mentoring me through some tough times and Dr. Derobertis for telling it like it is.

Closer to home, my office mates and friends deserve special mention: Dan Zingaro for B-ing Early to help me with my B troubles. Pouya Larjani deserves a medal for all of his explanations of logic.

Antoine Requet at ClearSy generously accepted all of my bug reports for Atelier B 4.0 and answered my questions about the environment.

Finally, special thanks to my Mom, Karen McLean. I couldn't have done it without you.

Contents

List of Figures

Chapter 1

Introduction

Jean-Raymond Abrial described modeling as taking the role of blueprint making for the field of software development [Abr09]. As blueprints allow one to reason about a car without actually building one, modeling allows one to reason about a program without creating it. Similarly a model can be refined by additional models as a blueprint may be refined by more specific ones. Initial models are very general to capture big ideas, then are refined in succeeding iterations adding greater functionality and data structures.

Behind many modeling systems including Atelier B 4.0, used here, is a formal mathematical logic used to generate proof obligations they must satisfy. B's logic system is based on classical logic and set theory and is understandable by anyone with a logic background.

B and UML are both modeling languages. What differentiates them is the underlying logic system in B that generates proof obligations (POs). A PO is a question to the modeler, "Have you considered this case?" Models satisfying all POs can be proven correct.

Modeling version control systems (VCSs) gives the community a set of common blueprints allowing communication in a programming language independent way about VCSs. New features can be created from the models and their impact examined without actual code being written.

The rest of this thesis is organized as follows: Chapter 2 introduces the Atelier B 4 modeling tool. Modeling version control systems literature is surveyed in Chapter 3. Particular attention is invested in a description of a UML meta-model of CVS and Subversion (SVN). Chapter 4 contains a description of VCS and an informal develop-

ment of SVN and CVS from available documentation. Development of initial models and their refinements is in Chapter 5. Conclusions and future work are in Chapter 6.

Chapter 2

The B-Method

2.1 Overview

B [Cle08, SK99, Abr96] is a formal method for creating models. A general overview of B and the features used in this thesis are presented.

Working in B one writes a specification (blueprint) of the project in an expressive first order logic with set theory in an *abstract machine*. It contains the model's variables: Integers, booleans, sets and relations amongst others. It's *invariants* describe properties that always hold true while *operations* change the state of the machine's variables. They are analogous to the variables, functions and operational rules in use in an actual program.

Correctness by design is enforced by the generation of *proof obligations* (POs) the abstract machine must fulfill. Each PO defines a condition of logical correctness or consistency that must be satisfied. Many are proved automatically. Some require the intervention of the user who must puzzle out whether the obligation can be met. If so, finding the proper pieces of information to feed the prover can be very time consuming and counter-intuitive. Often the prover generates a counter example. It demonstrates a refutation of the proof. In this case the abstract machine must be modified to account for it. Sometimes the prover cannot prove a correct statement. For large lambda relations the default search time must be increased or a hand proof performed.

An initial abstract machine is expected to be very general. In derived machines the data or algorithms are refined to more precisely reflect the item of study. *Data refinement* occurs when the variables are more precisely defined in terms of their

composition or the limits placed upon them: A *FileContents* variable refined to a sequence of integers, a set refined to an array. *Algorithmic refinement* occurs when the operations increase in complexity or number: A *descriptive* operation is refined by an *algorithmic* operation and operations that expand to handle data refinement.

2.1.1 Machine Clauses

A small example of a data queue from [SK99] illustrates many of the features of B used here. It manages a list of data items by maintaining a sequence of tokens referencing them. Free tokens are added, and existing tokens removed by the operations.

Listing 2.1: TokenQueue

MACHINE
DataQueue

SETS
$TOKEN$; $DATA$

VARIABLES
$AnyData$, $TokenSeq$, $TokenMap$

INVARIANT
$AnyData \in \mathbb{N} \wedge$
$TokenSeq \in \text{iseq}(TOKEN) \wedge$
$TokenMap \in TOKEN \nrightarrow DATA \wedge$
$\text{dom}(TokenMap) = USED$

INITIALISATION
$AnyData := 5 \parallel TokenSeq := [] \parallel TokenMap := \{\}$

OPERATIONS
$Success$, $AToken \longleftarrow AddItem(Item) =$
PRE
$Item \in DATA$
THEN
CHOICE

ANY *NewToken* **WHERE**
NewToken $\in$ *TOKEN* $-$ *USED*
THEN
TokenSeq := *TokenSeq* $\leftarrow$ *NewToken* ||
TokenMap(*NewToken*) := *Item* ||
Success := *TRUE* ||
AToken := *NewToken*
END
OR
Success := *FALSE* ||
AToken :$\in$ *TOKEN*
END
END;

DeleteItem(*Token*) =
PRE
Token $\in$ *TOKEN*
THEN
IF *Token* $\in$ *USED* **THEN**
ANY *Before*, *After* **WHERE**
Before $\in$ iseq (*TOKEN*) $\wedge$ *After* $\in$ iseq (*TOKEN*)
$\wedge$ *TokenSeq* = (*Before* $\frown$[*Token*] $\frown$*After*)
THEN
TokenSeq := *Before* $\frown$ *After* ||
TokenMap := {*Token*} $\lhd$ *TokenMap*
END
END
END

DEFINITIONS
USED == ran (*TokenSeq*)
END

A parent machine, if it exists, is found in the **INCLUDES** clauses. Variables are declared in the **VARIABLES** clause. *DataQueue* has three, that are typed in the **INVARIANT** clause. At a minimum the invariant section types all variables. It

often describes other constraints amongst the variables that always hold in the model, such as dom(*TokenMap*) = *USED*.

Variables are given initial values in the **INITIALISATION** clause. Note that a B sequence uses square brackets [], while sets use curly brackets {}.

Procedures are modeled in the **OPERATIONS** clause. They are descriptions in the sense that the actual program has to properly implement the functionality present in the model's operations.

Direct substitution macros are listed in the final clause, **DEFINITIONS**. As operations can't call other operations within the same machine, they serve a very useful role.

Within an operation the precondition block **PRE** contains a series of conditions that must be true before the body of an operation is executed. State is changed within the body. || indicates parallel composition. Each block of statements is assigned to one processor with no interprocessor communication allowed. At the end of each block the invariant must be reestablished.

Operations describe the functions and procedures of the model. Adding an item to the queue is descried by the *AddItem* operation. It has one input argument, *Item*, and two output, *Success* and *AToken*. *Item* is required to be from the *DATA* set. *AddItem* has two potential outcomes. If there are unused tokens left, a random one is associated with *Item* and added to the sequence. *Success* is set to true and the new token is returned. Alternatively, when all tokens are taken, *Success* is set to false and a random $:\in$ token is returned.

To remove an item from the set, call *DeleteItem*. It accepts a token already in the sequence. Within the body, *Before* and *After* are defined to be all sequence items before the token and after respectively. The input token is in neither list. Joining together *Before* and *After* completes the removal of the token from *TokenMap*.

2.1.2 Structuring Mechanisms

Different clauses provide different access rules when one machine inherits from another:

The most restrictive, **IMPORTS** gives machine B importing machine A access to the operations of A only. Variables in A cannot be accessed directly. They must be manipulated by referencing the operations of A.

Read only access to a parent machine is by **SEES**. Constants, state and query

operations are all accessible. Operations changing the state of the seen machine cannot be accessed, and the seeing machine cannot use seen machine variables in it's invariants, as another machine may change the seen machine's state, breaking the invariant.

Full access to methods is available through **INCLUDES**. Variables are not directly accessible, but can be manipulated by method calls in the included machine. This restriction allows each machine to be proved independently of it's parents and children. The child machine can relate it's invariant to the included machine. Because of the relation of invariants, a parent machine is only involved in one includes relationship. Allowing two or more creates the possibility of one child invalidating another child's related invariants.

2.1.3 Set Theory and Logic

The core of B is built around set-theoretic operations and logical connectives. Figure 2.1 lists the most commonly encountered ones. We assume familiarity with most of them.

Mapping a relationship from c to d is $c \mapsto d = (c, d)$. The direct product ($\otimes$) of relations $a \in (b \leftrightarrow c)$ and $d \in (b \leftrightarrow f)$ is a relation with elements $(g,(h,i))$ where $(g \mapsto h) \in a$ and $(g \mapsto i) \in d$.

2.1.4 Sequences

Sequences seq and injective sequences iseq are ordered listings of elements. They are total functions mapping the domain, $0..N$, $N \geq 0$ to a range, an arbitrary element type in a set. Injective sequences have the additional property that no element is repeated.

An empty sequence is written as []. Elements are [*e1*, *e2*, ..., *en*]. They are chosen using a functional notation: If s is a sequence, $s(n)$ retrieves the nth element $(0 \leq n \leq N)$. A summary of operations appears in Figure 2.2.

2.1.5 Correctness Criteria

Proving correctness requires all proof obligations to be discharged. They take three main forms: Initialization maintain the variables type integrity. Any operation adding or removing members from sets or relations maintains their type integrity. After any

$=$	equality
$\wedge$	conjunction
or	disjunction
$\neg$	negation
$\Rightarrow$	implication
$\forall$	universal quantifier
$\exists$	existential quantifier
$\in$	set membership
$\notin$	set exclusion
$\cup$	set union
$\cap$	set intersection
$\mathbb{P}$	power set
$\subseteq$	subset
card	set cardinality
closure	transitive reflexive closure
closure1	transitive (non-reflexive) closure
$\rightarrow$	total function
$\nrightarrow$	partial function
$\rightarrowtail$	total injection
$\mapsto$	maps to
$\lhd$	domain subtraction
$\otimes$	direct product
$\leftrightarrow$	relation
dom	domain of relation
ran	range of relation

Figure 2.1: Set-related and logic symbols.

first (s)	first element of sequence s
tail (s)	all but the first element of sequence s
last (s)	last element of sequence s
front (s)	all but the last element of sequence s
$s \leftarrow e$	sequence s appended with element e
$s \frown t$	concatenation sequences s and t
size (s)	number of elements in sequence s
$s \uparrow n$	first n elements of sequence s
$s \downarrow n$	sequence s with first n elements removed

Figure 2.2: Operations on sequences.

$$
\begin{array}{ll}
\text{WPA } x := E\,\{P\} & = \{P[E \,/\, x]\} \\
\text{WPM } x,\ y := E,\ F\,\{P\} & = \{P[E,\ F \,/\, x,\ y]\} \\
\text{WPI } \textbf{IF } E \textbf{ THEN } S \textbf{ END } \{P\} & = (E \Rightarrow S\,\{P\}) \wedge (\neg E \Rightarrow \{P\}) \\
\text{WPB } \textbf{BEGIN } S \textbf{ END } \{P\} & = S\,\{P\} \\
\text{WPS } S\,\{P\},\ \{P\}\ T & = S;\ T \\
\text{WPP } \textbf{PRE } Q \textbf{ THEN } S \textbf{ END } \{P\} & = (Q \wedge S)\ \{P\}
\end{array}
$$

Figure 2.3: Predicate transformers.

MACHINE N
CONSTANTS k
PROPERTIES B
VARIABLES v
INVARIANT I
INITIALISATION T
OPERATIONS
$y \longleftarrow op(x) =$
  PRE P
  THEN S
  END ;
DEFINITIONS D
END

Figure 2.4: Machine template.

operation on a variable, all invariants are maintained. All obligations the prover fails to establilsh are left as undischarged proof obligation for the user to struggle with.

Dijkstra's weakest preconditions are used to reason about a machine's state space and the transformations on it. For example, if $x := E$ is a B statement and P is a predicate characterizing a postcondition, $P[E \,/\, x]$ is the weakest precondition of $x := E$ to establish P. B substitutes all free x's in P with E. By composing these rules we generate more complex structures. Figure 2.3 is a list of weakest preconditions.

(Note that ; sequences B statements, where || parallelizes them.)

Figure 2.4 is a general template of a B machine. It's proof obligations (adapted from [Sch01]) are listed in Figure 2.5. Observe the final obligation in Figure 2.5 applies to all operations.

In Figure 2.5, (1) and (2) prove the static specification of the machine: Constants are typed and invariants are established. Rules (3) and (4) prove the dynamic consistency of the machine as it executes: Initial values of variables conform to the

(1) $(\exists k) \;.\; B$ – Constants can be instantianted

(2) $B \Rightarrow (\exists v) \;.\; I$ – Invariants can be established

(3) $B \Rightarrow [T]I$ – Initialization establishes the invariants

(4) $(B \wedge I \wedge P) \Rightarrow [S]I$ – All operations preserve the invariants

Figure 2.5: Proof obligations for template of Figure 2.4.

invariants and all operations maintain all invariants.

TokenQueue has no constants, so (1) is vacuously satisfied. For (2), any variable mentioned in the **VARIABLES** clause must be defined in the **INVARIANT** clause. This is true for *AnyData, TokenSeq and TokenMap.*

Rule (3) requires **INITIALISATION** to maintain invariants. Variable *AnyData* is initialized to 5. It is in $\mathbb{N}$ and preserves the invariant. [] (an empty sequence) and {} (empty set) are valid for *TokenSeq* and *TokenMap.*

Rule (4) requires invariants to be restored by the end of each operation. It's proof is nontrivial and not reproduced here.

2.1.6 Lambda Abstraction

B supports lambda abstractions. It is a formal system for defining and applying functions. In general they are recursive, though this behavior is not supported in Atelier B 4. In this thesis they are used primarily to iterate over sets of files presented to a VCS command. Their general structure is as follows:

λ A, B, ...({ *Type* A, B, ...}, {*Constrain* A, B, ...}
 |
 $C = F(A,\ B,\ ...)$)

The output of the lambda abstraction is of the type $((A \times B) \times ...) \nrightarrow C$. As an example we show a simple lambda abstraction that adds files from an external source to a file system.

SETS

$FILESSET$, $FILECONTENTS$

VARIABLES

$FileSystem$

INVARIANT

$FileSystem \in FILESSET \nrightarrow FILECONTENTS$

$ExternalFileSystem \in FILESSET \nrightarrow FILECONTENTS$

...

OPERATIONS

$OSAddFile(ExternalFiles)$

PRE

$ExternalFiles \subseteq FILESSET \wedge ExternalFiles \neq \{\}$

$\wedge\ ExternalFiles \subseteq \mathrm{dom}(ExternalFileSystem)$

$\wedge\ ExternalFiles \cap \mathrm{dom}(FileSystem) = \{\}$

THEN

$FileSystem := FileSystem \cup$

$\lambda\ AFile.(AFile \in FILESSET \wedge AFile \in ExternalFiles$

$\wedge\ AFile \notin \mathrm{dom}(FileSystem)$

$|$

$ExternalFileSystem(AFile))$

END

A straight forward union, $FileSystem := FileSystem \cup ExternalFiles$ is problematic because the same file name could exist in both relations. Adding it again violates the uniqueness of the domain of *FileSystem*. B generates an unsatisfied proof obligation for this case. In the lambda abstraction $\lambda\ AFile\ .(... \mid ExternalFileSystem(AFile))$, the output $AFile \mapsto ExternalFiles(AFile)$ is being added to *FileSystem* where $AFile \in FILESSET \wedge AFile \in ExternalFiles$ types and constrains the values *AFile* assumes. Guard $AFile \notin \mathrm{dom}(FileSystem)$ is necessary to prevent the same file name from appearing twice in the domain of *FileSystem*.

2.1.7 Supporting Software

Our model of SVN was developed using *Atelier B 4.0* B1, B2 and stable [Cle09] provided by ClearSy. It syntax- and type-checks B machines, generates POs and attempts to automatically prove as many as it can.

When the prover isn't able to find a rule to apply to a proof it cannot be auto-proven. For this the interactive prover helps us determine what went wrong. For example, sometimes a counterexample found by the prover breaks an invariant, or it finds a case where the output of a lambda relation is the empty set. One works with the prover by providing it hypotheses in an attempt to prove the PO (if possible.)

There are caveats we need to be aware of when using Atelier B 4: By default, expressions entered into the interactive prover are not syntax and type checked. If they fail either they are silently dropped. This is a problem as the mental model the user is working from diverges from the model Atelier B 4 has stored.

Within the main user interface, *proofs* and *lemma proofs* are stored in different, non-intuitive places in the project tree.

Using Atelier B 4's interactive prover is a black art. Many sacrifices of time and sanity will be made to the elder gods of logic as one works through a non-trivial machine. When an operation is nested within multiple *let*, *pre*, *if* and similar keywords it can lose track of type information. A simple reminder like adding a hypothesis, $ah(VerNo \in \mathbb{N})$ or $ah(VerNo \geq 0)$ is sometimes all that is required to satisfy a PO. There is little feedback from the interactive prover on what the problem could be. It is left to the user to puzzle out what could be going wrong and fix it.

One assumes the preconditions of an operation are stored and available to a PO. In practice this isn't done. They must be added as hypothesis.

Chapter 3

Formalities and Background

3.1 Version Control Classification

Version control systems (VCS) are widely used to manage collections of files and directories, along with the changes made to them over their lifetime. Any previously checked in version of a file is recoverable at any time from the repository.

VCS allow people to work on the same files in a decentralized and concurrent way, while consistently managing and integrating changes.

Using the classification schemes codified by Conradi [CW98] and Kipli [Kil97], CVS is a *file based control system* and SVN is a *file and directory based control system.* Changes to files are recorded as *directed deltas.* They use *extensional versioning*, where all versions are explicit and have been checked in before. All checked in versions are *immutable.* Their intent of evolution is *revision*, where each new revision supersedes it's predecessor. Version graphs are *directed acyclic* and support both *branching* and *merging.*

3.2 Version Control Terminology

Standard terminology is used when discussing VCS. Definitions are adapted from [Spi05a, Spi05b]. In general, files are understood to be, *files and directories.*

Under most circumstances a VCS is configured on a network. As it's primary goal is file protection and recovery, local storage isn't wise.[1] Users must be added to the permission list to access the repository, or more likely some subset of it.

[1] In multi-level VCS, one level of it may include local storage.

A file's life-cycle with respect to a VCS begins with an initial *check-in*. This tells the VCS to track this file and record changes to it when instructed. It is transferred to the repository and recorded as the initial version, or revision upon check-in.

After check-in the file receives a *version number*. It is 1 if each file has it's own version number (as in CVS,) or N ($N \geq 1$) if the virtual file system as a whole has a version number (SVN's model.) In the latter case, each check in increments the repository-wide version number by one and assigns this new number to all checked-in files containing changes.

Modifying a file requires it to be *checked-out*. A user downloads the file to their local system for modification. Binary files (images, sound, movies, many office formats like .odf) on both CVS and SVN require *locking* before check-out. Only one person works on a binary file at a time.

In some VCS, files are *cached on the client side*. A cache contains copies of the files in the state they were in when last checked-out, updated or reverted. They are not meant to be edited by users. Cached files allow a user to perform some operations even when the VCS isn't available, and can reduce network activity by allowing some operations to be done locally. One could determine the differences of a modified file with respect to it's cached or *pristine* state or undo changes made since the last check-out by copying over the file with the pristine file.

Moving, copying or *deleting* a file under version control is performed by invoking the operation from the VCS instead of the operating system (*svn mv myfile ...* instead of *mv myfile ....*) VCSs lose track of a file if it is moved, renamed or deleted using the supplied OS commands.

Before changes are checked-in, files are required to be *synchronized* with the server to receive any *updates* other users may have made to these same files. Changes to the version number of the file or the check-in date and time indicate another user has modified them and checked-in their changes. These changes from the server are *merged* into the user's copy of the file, bringing them up to date with the server. If user B has edited the same part of a file A has and committed, a conflict occurs when A updates. (See conflict below.)

Commit integrates changes made by a user on the local files into the VCS. Updated files receive a new version number as described above. Most VCSs can be configured to require a *commit message*: A short text explaining the changes, fixes, additions and/or deletions in the files.

When two users work on the same part of a file simultaneously and commit, a

conflict occurs. Jane's first commit works fine. Christa's is blocked, with a message informing her the part of the file she was working on has changed since she initially checked out the file. Christa must get together with Jane to discuss their incompatible changes and agree on a resolution, inform the VCS of this resolution (usually by editing specially prepared conflict files created by the VCS), then commit this new and integrated change.

VCSs don't resolve conflicts. They report their occurrence to their human overlords and leave it up to their social and personal skills to resolve their differences. One overlord may simply decide their approach is better and commit their change, effectively overwriting the first overlord's work. Where mediation fails, or never takes place, unresolved differences may lead to commit wars.

Commit wars are a series of needless, conflicting and mutually undoing changes committed by users who disagree on how a part of a file should look. Commit wars are sometimes started by (hostile) back-outs.

A *back-out* undoes the effects of a commit by either manually restoring the file to it's previous state and committing again or undoing a commit through the VCS using built-in administrator commands. A back-out is usually done by the initial committer. In contrast, a *hostile back-out* is performed by someone other than the original committer without any notice or prior arrangement.

Within a VCS the *head* is the most recent versions of all of the files on a particular branch. A *release* is a head made available to a wider community of users. Releases are often tagged as and begin a maintenance branch. The *trunk* is the main line of development of the items in question. Most branches originate from it.

A *branch* is set of VCS file versions identified by a common tag. They serve many purposes: *Maintenance* of older version of released items (eg: Bug fixing older code/program releases, corrections to the collection of recipes in a particular release of a cookbook.) Work on new features often occurs in *development branches*. When they're ready they are merged into another branch or the trunk. *Security branches* only allow security update and fixes to be checked in. Branches may be *frozen*. All work on them has ceased.

A *tag* is a name assigned to a specific release or branch. For example, Garden gnomes collection as of July 2009, V. 2.2 unstable, V. 2.1 stable or V. 2.0 maintenance.

Development of a branch ends when it is *collapsed* into another branch (usually the parent) or the trunk. From the parent's perspective, the child branch is *merged* into or *synchronized* with it.

Integration pulls changes from one branch into another. A collapse may or may not happen at the same time. *Re-parenting* a change is to move it from the source-branch to the destination-branch without committing those changes to the source branch.

3.3 CVS

CVS [Fou09] began as a collection of shell scripts created by Dick Grune. He posted them to comp.sources.unix in July 1986 [PGC08]. Brian Berliner created CVS in April 1989. Jeff Polk helped with the CVS module and vendor branch support. It is available from their web site at [Fou09].

References to CVS are to the user manual version 1.11.23.

3.3.1 Informal Description of CVS

CVS manages files and changes made to them over time. Within the CVS repository the directory structure mirrors the directory structure of the local files when they're initially checked in. Every file under version control is stored in it's own RCS file [2]. It contains the full file contents from the initial check-in and stores subsequent updates as a series of change deltas describing the changes made, who made them, when and why.

Writing changes to files as deltas is more efficient than recording the full file on each check-in. Any previous version of the file is recoverable by retrieving the initial check-in followed by applying the deltas in order until the version is reached the user is interested in.

CVS uses the *copy-modify-merge* (CMM) model for text files and the *lock-modify-unlock* (LMU) model for binary files. In CVS-speak, text files use *unreserved checkouts* while binary files use *file locking*. Users must remember which files are binary and specify so using command line parameters. Forgetting means the file is treated as text, causing corruption. As CVS doesn't work well with binary files, the *full file* must be recorded every time a change is checked in. Change deltas can't be used.

CVS supports branching and tagging, but has difficulty with binary files on branch merges. Merging changes from one binary file into another is meaningless and corrupts the resulting file.

[2] RCS files come from the RCS VCS, a predecessor of CVS.

CVS doesn't support transactions. When a directory of files is checked-in, CVS locks each corresponding file in the server in turn, allowing it's deltas to be stored without interference from others. Should Bill check-out while Claire is checking-in, Bill receives some updated files from Claire and some files yet to be updated by her. Should Bill attempt to check-out a file in the middle of an update, the request is denied because it is locked by Claire's update.

A group of CVS file ends up in an inconsistent state if a commit fails part way through. Changes will be applied to some files, but not others. If a file is partially modified when the commit fails, it is ignored. There is no way to roll back an incomplete commit.

Conflict resolution is weak. A user is under no obligation to resolve the conflict with their fellow human beings. CVS allows someone to check-in regardless, initiating a hostile back-out. Users should always discuss and resolve their conflicts before checking-in. CVS is of no help here.

CVS handles network traffic efficiently for text files. When a user updates a local file to the CVS server, only changes to the file since that user checked it out are sent. Delta packets are used and applied at the client end. When checking-in a file, it is sent in full - and may be compressed. The CVS server calculates the delta packets and appends them as a new revision to the RCS file.

Within CVS, every file has it's own version number. It starts at 1 and increments by 1 on every successful check-in.

CVS doesn't support file moving, file renaming and directory moving and renaming. Any client operation moving or renames a file creates a broken history. After the operation the history of the file is still associated with it's *old-name*. As far as CVS is concerned, *new-name is a new file with no history*. No link is maintained between new-name and old-name. A user is expected to remember all operations causing file-name changes, and what version number this change occurred at.[3] A *CVS Wizard* can apply their *eldritch sorcery* to the server to attempt to re-link file histories.

Renaming a client directory requires four separate steps: Individual files are moved out of the old directory, causing broken histories. The old directory is deleted and the new one created. Finally the files are moved into the new directory and added to CVS as files newly placed under version control.

Repository directory permissions on Unix must be managed by creating groups with the appropriate read and write access on the directories of the CVS server. Users

[3]You just started working at the company today, and the CVS guru retired yesterday?

must be added to the appropriate groups, which may require the intervention of a Unix system administrator. The granularity of control is limited to the directory level. Individual file control must be managed by splitting them between directories. CVS assumes subdirectories inherit their parent's permissions.

3.4 Subversion

Subversion (SVN) [Col09] was specifically created to remedy flaws in CVS. By design it is similar to CVS but with a more modern and complete feature set.

3.4.1 Informal Description of Subversion

Information on SVN is drawn from web documentation at [CSFP07].

SVN users start to notice the differences between SVN and CVS when they add their first project to it. Any time a file is checked out, a *cached* or *pristine* copy of each file is always stored as well. The designers reason disk space is cheap, where network bandwidth isn't. Always having a pristine copy of the checked out files allows more operations to be performed without the involvement of the network and server.

Storage and transfers are built around a binary difference algorithm. It erases the difficulties in using binary files. When a file is first imported or added, a binary detection algorithm is run on it. From this the *svn:mime-type* is set to *text* or *binary*, alleviating the user from the responsibility of remembering.

SVN is designed to treat binary and text files the same—as literal byte strings. No keyword or end of line translation is performed, like in CVS. As the file type is accounted for end-to-end and a pristine copy of the file always exists on the client side, change deltas are sent both ways for both the text and binary file types.

Within the SVN repository it is possible to record the changes to a binary file with a series of space saving change deltas.

Files and directories are versioned in SVN. Within the server, the *versioned entity is the file system*, not the individual file. Revision numbers apply to these file systems. Every revision is a snapshot of the file system after a check-in. In CVS it's guaranteed the contents of versions n and $n + 1$ of a file are different. Within SVN a file or directory may not change over an arbitrary number of snapshots.

As the base entity on the SVN server is a virtual file system, file and directory deletions and renaming are all tracked. There are no broken histories. If a file or

directory is renamed using SVN and checked in, the newly named file has a link to it's previous history. No file copying or necromancy is required.

A *tree delta* describes the changes to the directory tree from one version to the next: The copies, renames and deletions of files and directories and changes to file content.

As expected, SVN uses the *copy-modify-merge* model for text files and the *lock-modify-unlock* model for binary files. Files are checked-out, edited and updated as expected. *Ancestry sets* remember what changes have been merged into files when other user's changes are incorporated into the local files on update. This lowers network usage by insuring changes aren't incorporated into client files more than once.

SVN requires any conflicts between updates from the server and a user's edits to be resolved before check-in. It creates a specially marked up file showing both overlapping edits. The file cannot be checked in until the markers are removed and some choice made about the contents. *SVN resolved* must be explicitly run to apply the update.

A conflict must be resolved, but still doesn't enforce human cooperation. (No system can.) Hostile backouts still occur. Unlike CVS, it can't happen by mistake.

SVN was designed with a number of database principles in mind: Any repository operation modifying files must first acquire an *exclusive write lock* on them all, and must wait if one or more are already locked. Once it has the locks, it performs the actual operation. If there is some kind of error, such as not receiving all the data, the operation is *undone* and an error message generated. Either all of the operations take place, or none do.

Serialization avoids the problem of lost writes or partial writes. If two separate users update the same file at the same time, locking and atomicity guarantee that one update will occur, then the other. It doesn't guarantee any order, only that they happen in turn.

File system space and branch space are the same in SVN. They are all ordinary directories in the virtual file system under the root directory of a project.

3.5 Differences between SVN and CVS

Comparisons between SVN and CVS have been made numerous times before. This list was drawn in part from [WG05].

SVN	CVS
SVN uses the revision number to determine when the client files are out of date with the server and time stamp to know when the client file is changed compared to the pristine copy.	CVS client looks at the time stamp of the file to determine whether a check-in is needed. The server looks at actual file contents.
There is one global revision number per virtual file system tree.	Each file has its own revision number.
From revision to revision, a file might not change.	From revision to revision, a file changes.
Files, directories, copying and renaming are all under version control.	Files are under version control, directories are not. Copying and renaming are not supported: File history will be broken.
Stores pristine copies of files locally. This allows more off-line operations like status, diff and revert.	
Subversion doesn't allow conflicted files to be accidentally merged into the repository. Hostile backouts can be performed.	Conflicted files can be accidentally and intentionally submitted to the repository. Hostile backouts can be performed.
Uses a binary differencing algorithm, allowing changes to be stored as delta packets. Said delta packets are transmitted both ways: Client to server and server to client.	Clients send full files on every update. Server sends text delta packets or full binary files to the client.
Supports transactions and atomic commits. Can rollback incomplete transactions.	Supports atomic commits of single files only. No transaction support, no rollback support.

Arbitrary *name* = *value* metadata can be attached to files and directories and is versioned.	Tags are supported for version identification and branching.
Uses mime type (RFC 2046) to determine if and remember whether a file is binary or text.	Users must remember to flag binary files. If forgotten they are sent as text with endline characters inserted and are garbled.

Table 3.1: Subset of differences between SVN and CVS

3.6 Choosing Features to Model

Our goal is to decide on a set of features to model and carefully define them. Commands should be chosen that cover the major functionality of CVS and SVN. The most commonly used ones include *add, delete, move, copy, update, revert, check-in, check-out* and *status*. Administrator and configuration commands were excluded as they are too implementation dependent.

As VCS store more than files, we use the more general term, *item*. For CVS item encompasses text and binary files. When discussing SVN item encompasses text and binary files, links and directories.

Add: [Item] Schedules items to be added to the repository on next commit. **CVS Notes**: CVS doesn't recurse into sub-directories. Binary files must be specified. **SVN Notes**: SVN recurses into sub-directories and seamlessly supports binary files.
Delete [Item] Deletes local items immediately and schedules them to be removed from VCS management on the next commit. Histories of deleted items are preserved in the repository. Items not under version control are left alone. **CVS Notes**: User must issue separate operating system commands to delete local items. **SVN Notes**: Local items are deleted automagically.

Copy [Source] [Destination]
Copies items from source path to destination path immediately locally and schedules the newly created items for addition upon the next commit.
CVS Notes: CVS doesn't directly support copy. Instead the following must be done:
cp old new (History broken here.)
cvs add new
cvs commit -m "Copied old to new" old new
New files start off with a version number of 1 unless specified otherwise. Item histories are in two places now: *Old* for everything before the copy and *new* for everything after. Users are expected to know about and remember all copy operations. To copy a directory, create the destination directory then perform the copy procedure above for each item in the directory.
SVN Notes: Copy preserves file histories by linking to the source file. Directories are copyable with a single command.

Move [Source] [Destination]
Moves items immediately locally and schedules the same for the server upon the next commit.
CVS Notes: CVS doesn't have direct move support. Instead the following must be done:
mv old new (History broken here.)
cvs remove old
cvs add new
cvs commit -m "Moved old to new" old new
New files start off with a version number of 1 unless specified otherwise. item histories are in two places now: *Old* for everything before the move and *new* for everything after. Users are expected to know about and remember all move operations.
To move a directory, perform the move procedure above for each item in the directory, then delete the directory as described under delete.
SVN Notes: Move preserves file histories by linking to the source file. Directories are movable with a single command.

Checkout [Item]
Copy items from the repository to your local system.
CVS Notes: Users must remember and specify whether an item is binary or not. Getting this wrong mangles the client-side file.
SVN Notes: SVN's binary detection algorithm allows it to seamlessly handle binary files.

Update [Item]
Brings a user's working copy into sync with the latest versions in the repository by integrating changes others have committed to the server.

Commit [Item]
Store changes in items back to the repository.
CVS Notes: Users must specify which files are binary.
SVN Notes: SVN remembers which files are binary.

Revert [Item]
Undoes any changes made to the specified items.
CVS Notes: CVS doesn't have a revert command. *cvs update -C [item]* downloads the latest copy of the item from the server. It isn't necessarily the same file as before as changes made and checked-in by other users are included. (Alternatively one may checkout the now older version to effectively revert.)
SVN Notes: Items are reverted to the pristine states stored locally. This allows disconnected reverts. Reverts on items cannot be undone, but item additions and deletions can.

Status [Item]
Shows the status of checked out items with respect to the pristine copy (SVN) or the server.
CVS Notes: There is no specific status command. Update, history, and checkout commands (amongst others) give the status of a item with respect to the CVS server:

Status	Meaning
U	Item has been brought up to date with the server by sending the full file to the client.
P	Like U, but only file deltas (a patch) were sent to the client.

A	Item is newly added to the VCS and is awaiting it's first check-in.
R	Item has been deleted by the client and will be removed from version control on the next check-in.
M	1) Item has been modified locally and the repository version hasn't been modified. 2) Item has been modified locally and updates from a new server version have been merged into item without difficulty.
C	Item is in conflict with the latest version on the server due to overlapping edits.
N	There are no changes to the file.

Observe there are no copy or move statuses.

SVN Notes: Differences between SVN and CVS are shown in the following table:

Status	Meaning
D	Item has been deleted by the client and will be removed from version control on the next check-in. Item history isn't affected.
R	Item has been deleted by the client and was then replaced by another file with the same name.
!	Item is missing. It could have been moved or deleted by a non-SVN command for example.
~	Item has been replaced by a different type of object (directory, link or file).

In SVN, moving a file is represented as an addition followed by a deletion:

svn move PieRecipe.txt CakeRecipe.txt

A CakeRecipe.txt

D PieRecipe.txt

3.7 Environmental, Functional and Safety Requirements

Here we list rules describing CVS and SVN models. They describe the system in a more systematic way. Amongst them will be candidate invariants, preconditions and postconditions usable in the B model. These rules are divided into three categories: Environmental, functional and safety.

3.7.1 Environmental Requirements

1. VCS are software systems consisting of multiple parts: Client and server systems and software, network connections, client and server VCS software, client items under and not under version control and the server repository.
2. VCS are multi-user systems.
3. VCS work transparently across different operating systems with different network types, file systems, binaries and end of line markers.
4. VCS servers may be located on the same computer as the client, but are much more likely on another computer accessible through a network.
5. Items under version control and not under version control can exist in the same client directory.
6. SVN has a pristine cache on the client computer. It is located in the hidden .SVN directory in every directory under version control. Users should not manipulate it's contents.
7. SVN offers limited functionality through the pristine cache when the server isn't available.

3.7.2 Functional Requirements

Basics

8. VCSs run forever and should always operate (and fail) correctly.
9. A VCS maintains a complete history of all changes to items under version control starting from their initial check-in.

10. In CVS, text and binary files are versioned. In SVN, text and binary files, directories and links are versioned.

11. When initially created, the repository is empty and no items are under version control.

12. Item(s) are placed under version control by informing the VC system about them using an add or import command.

13. CVS supports item tagging for the identification of releases and branches.

14. SVN supports arbitrary *name* = *value* metadata for items. Said metadata is also fully under version control.

15. CVS supports file locking on the server for updates, but not transactions.

16. SVN uses item locking, transactions and the multiple reader/single writer model for all transactions against the server. This orders concurrent operations from multiple users in an unspecified way and guarantees partial reads and writes will not occur.

17. A SVN transaction succeeds after it has been fully collected by the server and applied to the corresponding items. It fails and is undone otherwise.

CVS: Version Numbers, Checkout and Update

18. Every file under version control has it's own version number.

19. A file's version number increases by one on each successful check-in.

20. A version is recorded only if there are changes.

21. After check-out the client item's version number is equal to the items version number in the repository.

22. Other users may check in changes to items, increasing those items version number. When changes are downloaded from the repository, all changed items contents and version numbers are updated to the values from the server Repository.

23. As individual items of a client project may be updated and checked in, version numbers of client items fall between A and B, ($A \leq B$; A, $B > 0$) where A is the minimum of the version numbers of items that have not yet been updated and B is the maximum of the version numbers of updated items.

24. For a check-in to succeed, there must be changes in the item. The change delta between version N and N+1 for an item may not be null.

25. As a consequence of 7, for file version number $N > 0$, all versions $1..N$ are found in the repository.

26. Once a version is recorded in the repository, it can never be changed or deleted by any operation. It is forever immutable.

SVN: Version Numbers, Checkout and Update

27. An entire repository has a single global version counter. It is set to 1 when it is initially created.

28. The global version number of the repository increases by 1 only when items are checked in.

29. At least one item in a check-in transaction has to have changed for it to occur.

30. Server items have version number equal to or larger than the client and cache.

31. Pristine items have a version number equal to the server version number at checkout. Only the checkout, update and revert operations change the pristine items contents and version number.

32. After checkout the client item's version number is equal to the server version number and the cached version number.

33. Other users may check in changes to items, increasing the server version number. When changes are downloaded from the repository, all changed items (in both the client directories and cache area) contents and version numbers are updated to the latest from the server.

34. As individual items of a client project may be updated and checked in, version numbers of client items fall between A and B, ($A \leq B$; A, $B > 0$) where A is

the minimum version number of the cached items at the last checkout and B is the maximum version number of the checked out items.

35. After items are checked in, there may or may not be changes to them. Regardless, all item version numbers are set to the global version number + 1. This means the change delta between version N and $N+A$ (N, $A \geq 1$) for an item may be null — if that item wasn't modified in that interval.

36. All version numbers of all revisions of items are less than or equal to the global version number.

37. As a consequence of 3, for global version number $N > 0$, all of $1..N$ are found as version numbers in the repository. Each version number appears a minimum once and up to Z times, where Z is the number of items under version control.

38. Once a version is recorded, it can never be changed or deleted by any operation. It is forever immutable.

CVS: Server

39. Every file under VC has a corresponding file within the repository.

40. Upon initial check-in the entire file is sent from the client to the server to be recorded.

41. Changes to text files are stored as change deltas: Only differences between revisions are recorded, not the full item.

42. On check-in, full binary files are recorded. Deltas are not used.

43. Full files are sent from the client to the server. Change deltas are sent from the server to the client for text files, and full binary files are sent from the server to the client.

SVN: Server

44. Every version can be thought of as a directory tree recording changes between it and the state of the tree of the previous check-in.

45. All changes are stored as deltas for both text and binary files: Only differences between revisions are recorded, not the full item.

46. Change deltas are sent both ways over the network. SVN's designers reason that hard disk space is cheap, while networking capacity is expensive.

Add, Delete, Move, Copy

47. VCS only manipulates items placed under version control. Everything else is ignored.

48. CVS must be informed if an item is a binary type each time it is manipulated. Incorrect specification causes it to be treated as text, adding end of line markers and ruining it.

49. SVN uses a binary detection algorithm to determine if a file is a text or binary type. It records this information for all future transactions.

50. VCS must be informed about local changes to items under version control. Adds, deletes, moves and copies must be done through the appropriate VCS commands.

51. CVS doesn't support move and copy. Moving or copying a file causes a broken history in the repository. Moving or copying a directory causes broken histories for all child files of the directory.

52. Humans are expected to remember all moved and copied file's original names to be able to retrieve those histories from the repository.

53. In SVN, when a item is copied, moved or renamed, links are maintained to the previous item and metadata history on the server. Item history is neither lost nor duplicated.

Merging, Conflicts and Those Pesky Humans

54. Jane may have modified an item and checked it in while Bethany is still working on it. If Bethany tries to check in, she learns her items are out of date. She must merge Jane's changes into her items before checking in.

55. If Jane's changes don't overlap Bethany's, merges are automatic. Bethany may now check in her changes.

56. If Jane's and Bethany's changes overlap, the VCS rejects Bethany's changes. Said VCS stops with an error message and marks up the items in question with information about the conflicting area(s).

57. VCSs have no ability to resolve conflicts. They only see items as streams of bytes.

58. Human intervention (And hopefully communication) is required to resolve all conflicts. The conflicted items must be edited, a choice made and the VCS informed before said items can be updated.

59. One user can resolve the conflict without consulting any others. This is allowed for cases where the other party isn't available. Conflicts this may cause, like revert wars, must be resolved by the humans involved.

60. In CVS, a conflicted item may be checked-in by mistake. This isn't possible in SVN.

61. For SVN, conflicts only occur in text and binary files. Links and directories cannot be conflicted.

Reverting and Retrieving Previous Versions

62. In CVS changes to client items are undone by reverting them from the server.

63. In SVN changes to client items are undone by reverting them. Items are replaced by the same item from the client cache, or the server if an older copy is desired.

64. SVN allows a user to check the status of their file with respect to the pristine cache or server. CVS only checks against the server.

65. If a user retrieves an older version of a item, changes it and checks it in again, that older version in the repository isn't changed. As described above, version history is immutable. The change is given a new version number and checked in as the most recent revision.

3.7.3 Safety Requirements

66. Locking is used for binary items.

67. Locks can be broken by third parties when necessary.

68. SVN: Should the client, server or connection fail in the middle of an operation, the system state upon restoration should be that of the operation never occurring. Incomplete operations are rolled back.

69. Operations with missing or malformed parameters or arguments fail gracefully with a helpful error message.

70. SVN: Some operations like diff and revert can be performed against the client cache. The presence or absence of the server is irrelevant.

71. SVN: Operations against a corrupt or missing cache or server fail gracefully with an appropriate error message. Atomicity guarantees that when an operation fails nothing changes.

72. SVN: A missing or corrupt cache is repaired by checking out the items again.

3.8 Refinement Steps

Refinement	Features
1st	Operations defined, arguments and return values are typed.
2nd	Client item system, VC repository and SVN global version number.
3rd	Version number of checked-out items.
4th	List of items under version control server-side.
5th	List of items under version control client-side.
6th	Client-side shadow repository.
7th	Status operation.
8th	Binary item support.
9th	SVN only: Pristine cache.

Atelier B 4.0 is used to model CVS and SVN in a series of algorithmic and data refinements. For algorithmic refinements the abstract and concrete state spaces are the same. Only control structures may change. In data refinements the abstract and concrete state spaces are different.

Our base model is similar to a virtual class in that it is a machine which defines operations, the types of their arguments and return types for both SVN and CVS. Guards and bodies are empty. Note that Atelier B 4 does not allow empty operation bodies. To overcome this the *skip* operation is used. It should be interpreted as doing nothing.

Operations *add, delete, move, copy, commit, update, checkout* and *revert* are specified for SVN. CVS is similar, but doesn't specify move and copy. Add, delete, copy and move are further divided into client and server operations: *Add, commitadd, copy, commitcopy, move, commitmove, delete* and *commitdelete.*

Conceptually client operations are performed immediately. *Add* for example adds items to a queue to be placed under version control. *CommitAdd* is executed when commit is called. It performs the rest of the add operation on the server side.

Environmentally both the SVN and CVS models consists of a single client with an abstracted item system where multiple items are under version control. There is no concept of directories, users and permissions in the item system. There is an unspecified network separating the client from the server.

The second model defines the client item system and the VC repository. SVN versions directory trees instead of individual items, so it adds a global version number counter. Note that some operations are empty or missing. Operation *Add* for example has nothing to do as there is no client side list of items under version control. *CommitDelete* is also empty as items in the repository are never modified or deleted.

Tracking of the version number of checked out items is added in the third refinement. As items are copied, moved, deleted, checked-out and checked-in this relation must be constantly updated. For example, the item created by a copy operation is added to the checked-out list upon check-in. Note that once an item version is recorded in the repository it is immutable. It is not possible to delete items from the repository when *CommitDelete* is called.

Keeping an explicit list of the items under version control is added in the fourth refinement. The version number of a checked-out item and the list of items under version control must necessarily be separate because a item that is not checked out is still under version control. A item resulting from the copying of a checked out item

is both under version control and checked out.

The fifth refinement tracks items under version control on the client side. It is a shadow that keeps track of the status of a item before it is checked in. This allows multiple operations on items to be tracked. One can can arbitrarily add, delete, copy and move items before invoking check-in and *commitcopy*, *commitmove*, *commitadd* and *commitdelete*.

Keeping track of which items are under version control on both the server and the client is important due to the split nature of the add, delete, move and copy operations. Calling *Delete* say removes items from the client version control list. Correspondingly the server list isn't affected until *CommitDelete* is called.

In the sixth machine a shadow repository is added. It is a exactly like the server repository in the first model: Once an item is placed in it, it cannot be modified or deleted. Like the fifth machine it is also a shadow. It records all changes to client items from client operations (Add, delete, move, copy) and makes them available to the corresponding server operations. For SVN a shadow version number counter is also present.

Machine seven adds a status operation. It determines the status of an item with respect to the server. Values include: Newly added, deleted, moved, copied, out of date and up to date. The first four means the client operations add, delete, move or copy have been called and the corresponding server operations have not. Out of date means the item has been modified and checked-in by another user in the meantime.

Iteration eight adds binary item support. For SVN it's automatic. In CVS a user argument is added to specify whether a item is binary or not. Forgetting to specify a item as binary and specifying a text item as binary in CVS is treated as an error.

Revision nine add the pristine cache for SVN. This saves the item locally in the state it was in upon check-out. This allows some operations to be performed without the server being available. For example, one can revert to the pristine copy at any time or compare an edited item to the pristine. CVS doesn't have a pristine cache so it doesn't have a ninth revision.

Chapter 4

Related Work

No previous work was found on the topic of modeling VCS. Morgan and Sufrin presented a paper [MS84] where they derive and refine a specification of the Unix level 6 file system using elementary mathematical set theory. Leo Freitas et. al. presented a paper [FFW09] where they created a formally verified specification of a POSIX-compliant file store using Z/Eves. Both are related as VCS are intimately involved in file transfer and manipulation. Even though the file system is abstracted here, reading both of these papers is very valuable for their contributions and insights.

Marjanovic's thesis [Mar06], Meta-Models VCS and Issue Tracking Software (ITS) to create a model of Release History Systems (RHS) is related. Meta-modeling is described as:

> The application of valid frameworks to describe the semantics of different conceptual worlds on different abstraction levels, whereby it is possible to layer the frameworks used for description.

His thesis begins by developing a meta-model of VCS in UML 2.0 [Gro09] capturing the features of both CVS and SVN. It is extended by the Bugzilla issue tracking system to create a meta-model of a RHS. Validation of it's suitability is performed against Rational ClearCase and it is implemented in Java with Hibernate.

4.1 Meta-Modeling of VCS Using CVS and SVN

Our interest is in the first part of [Mar06]: Meta-modeling VCS by examining CVS and SVN. It is shown in detail as it provides a complimentary and visual representation of the more terse and arcane B model.

Marjanovic's purpose is to find a common meta-model to encapsulate both CVS and SVN. Data fields are determined by examining the server. They are divided into entities and related to each other by examining their use in different client software. Keeping their purpose in mind, a number of errors, redundancies and vague entries exist in both models. The original *Entity — Relation diagrams* (ER) [Mar06] for SVN and CVS are shown and their problems described. New ER diagrams are derived for each to suit the purposes of this thesis: A visual representation of CVS and SVN to complement the B models.

4.1.1 Initial CVS Model

In Figure 4.1, *CVS-Entry* is the entity describing a file under version control on the CVS server[1]. *RCS file* is the path to it's corresponding RCS file containing it's change history. *Working file* is the name of the checked-out file. *Head* is the most recent revision of a file. As the last entry in the RCS file is the head, it is redundant. *Branch* is also redundant, as it is described in a separate entity.

Locks is for reserved checkouts. *Access list* is described as, "The permitted user list." The only reference to access lists in [PGC08] is in section A.7.1 where they are described as something available in the RCS file, but not used by CVS. Manipulating this access list could cause CVS to stop working. Access to files in CVS is controlled by permissions on groups in directories on the server. It is removed.

Symbolic names refers to the tags associated with a file version. Tags are used to mark files as being part of branches and releases. It is redundant here as branches and releases are full entities in their own rights. *Keysub* has no description. It could be the keyword substitution mode (Section 12.4 of [PGC08]) or a list of keyword substitutions maintained by CVS. We believe it's the latter[2]: Listing them in each file is redundant. They are better expressed in a singleton Keyword entity.

Entity *CVS_Entry_metainfo* is redundant. Release and branch tags are entities in their own right and keywords are expressed in a singleton Keyword entity.

ModRep is short for modification report. It details what changed upon check-in of said revision. *Revision* is redundant with the Revision entity, as branch is with the branch entity. *Date*, *author* and *log message* are as expected: Who checked-in, when and a note describing what they did. *State* and *lines* are poorly described.

[1] CVS doesn't version directories.

[2] Keyword substitution mode is far too specific a property to model.

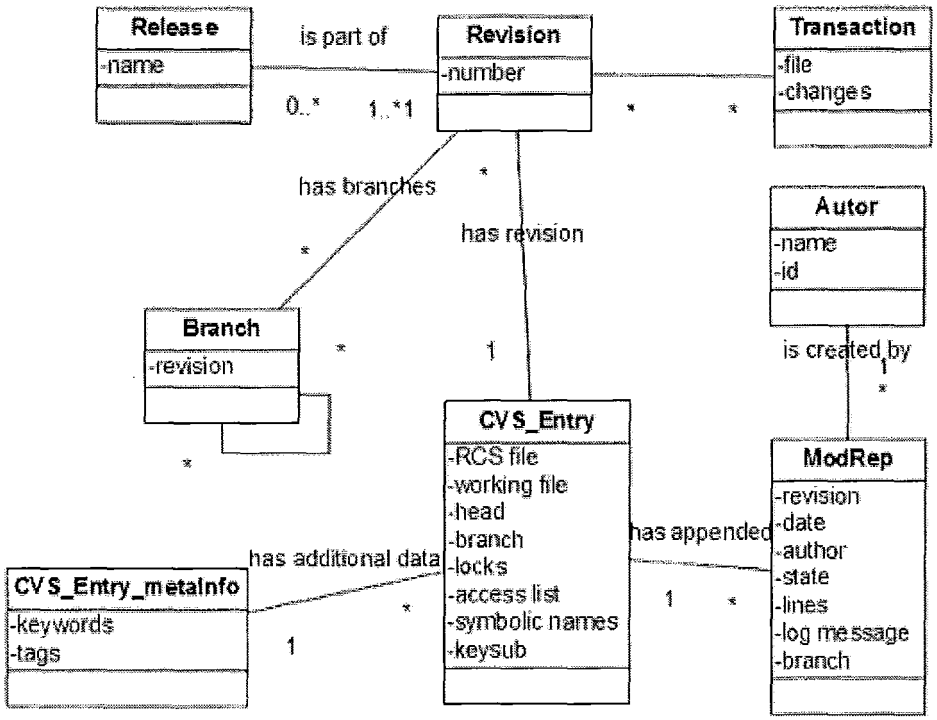

Figure 4.1: UML model of CVS from [Mar06].

It's reasonable to assume they encapsulate changes made to the file along with other persistent state information.

This entity will be merged into revision, to better encapsulate what changes.

Autor (sic) is the author entity. With ModRep absorbed into Revision, it also links to Revision.

Revision is the current revision number of a RCS file. It is of prime importance and is separated from all other entities.

Release is a collection of file revisions with a common release tag.

Branch is a collection of tagged files of a certain revisions split off from the parent revision branch to their own development branch. A revision can have multiple branches. A branch can have multiple revisions. Branches can recursively split from branches.

Transactions are not part of CVS. An ongoing effort at the University of Zurich Department of Informatics, Software Evolution and Architecture Lab, is working on adding them. As it is not in CVS as of 2009, this entity is removed.

Confusion over the number of entities involved in the Release-Revision and

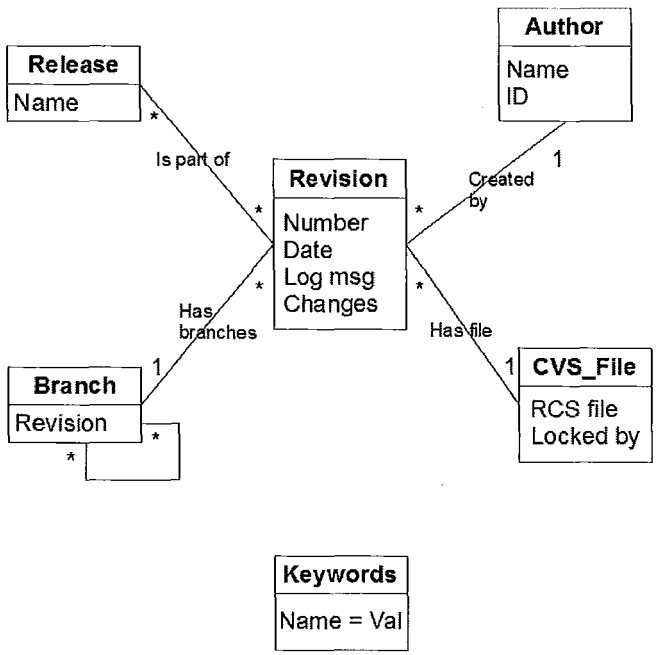

Figure 4.2: Revised UML model of CVS.

Branch-Revision relationship is clarified.

4.1.2 Revised CVS Model

Figure 4.2 captures the essence of the relations between entities. It shows the file based nature of CVS in the entity name, *CVS-File*. The relation reinforces this: A revision consists of a single file.

4.1.3 Initial SVN Model

As SVN is intentionally similar to CVS, Figure 4.3 is very similar to Figure 4.1. *SVN File* is the entity describing both files and directories under version control. URL is, as expected, the URL of the file within the repository. *Revision* is a redundant property of (the missing revision property in) the Revision entity[3]. *Author* is described as,

[3]This sentence may need revision.

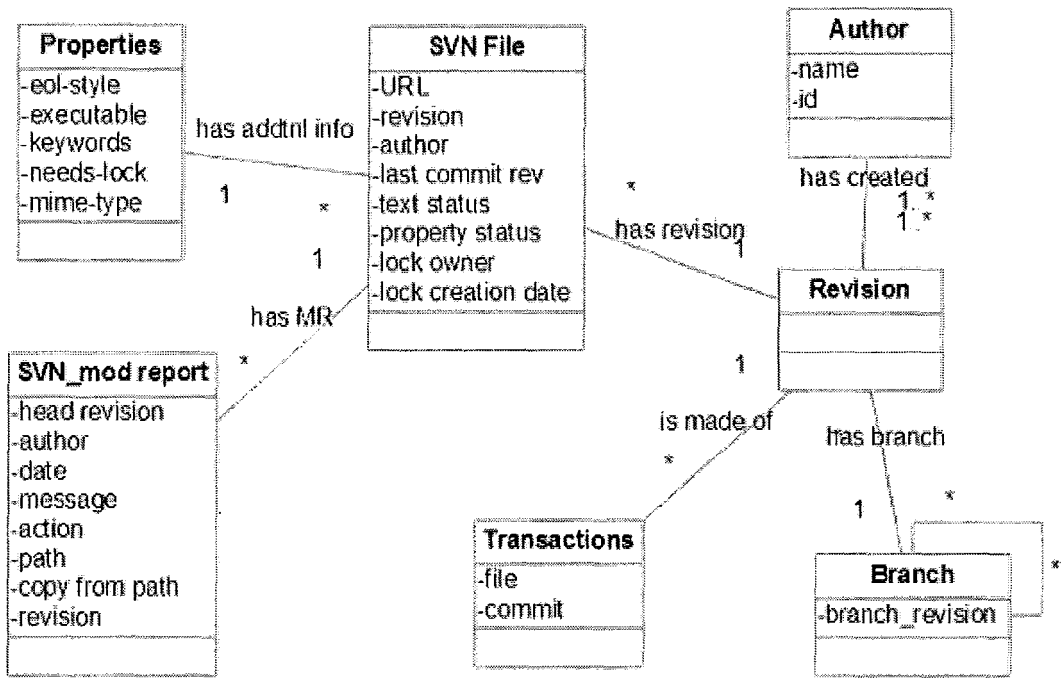

Figure 4.3: UML model of SVN from [Mar06].

"The author to whom the file belongs." It is unclear if this is the user who did the initial check-in, users who have permissions on the directory containing the file, or the last known user to change the file ([CSFP07], pg. 53). We assume it is a redundant field describing the author of the revision and delete it. *Last commit rev* is redundantly redundant to the redundant property above and is also deleted.

Text status describes the status of the checked out file with respect to the repository. It's values include: Modified locally, modified in the repository, added, deleted, copied and moved amongst others.

Property status describes non-versioned properties of a revision. An example is the check-in date. This conflicts with the definition of properties in the manual ([CSFP07], pg. 40). It describes properties and meta-data as versioned "name=value" pairs, with only a passing mention of non-versioned properties. We assume properties covers both versioned and non-versioned items.

Lock owner and *lock creation date* have the expected meanings.

SVN-mod report is SVN's modification report describing changes upon check-in. *Head revision*, *author* and *revision* are all redundant. *Date* and user *message* are as

expected. *Action* isn't described. We assume it's the action performed by the item upon check-in: Added, copied, deleted, renamed or moved.

Copy from path also has no description. We assume it's the source path when the action is copying, renaming or moving an item. [4] Similarly the not-described *path* is assumed to be the destination name for a copy, move or rename.

Date and *message* are moved into the revision entity.

Properties is additional information stored in the metadata and tags associated with an item. *Eol-style* is the end of line style to use on checkout of textual data. As SVN detects both the OS and file type, the default end of line style for text files is *native* — what is expected for that operating system. There is no eol-style for binary files. *Executable* describes whether or not a file is an executable on file-systems who have execution bits. *Keywords* are defined within SVN and should be in a singleton entity.

SVN tries to automatically determine the type of an item on check-in and sets the *mime-type* to either text for text files or application/octet-stream for binary data. *Needs lock* is true for binary data and false for text. Observe that both eol-style and needs lock are determined by the mime-type and are redundant. Text files (such as Unix scripts) may be executable, so the executable property must stay.

For the *Transaction* entity, [Mar06] suggest a transaction applies to a single item only. It applies to any number of checked-out files and directories. *Commit* isn't described at all.

As the transaction entity is so poorly specified, it is deleted. There is no field in any of the entities describing changes to items from one check-in to the next. Two are added.

Tree-changes is a field added to the revision entity. Conceptually it describes all of the changes in the directory tree from one revision to the next. *Delta-packet* describes the changes from one revision to the next for an item. As items don't have to change from one revision to the next, it can be empty.

Branch and author are the same as in CVS.

4.1.4 Revised SVN Model

Figure 4.4 captures the properties of SVN. Transaction support is implicitly reflected by the *Revision - SVN_Item relation.* A revision consists of one or more items.

[4] SVN preserves an item's history upon rename, copy and move.

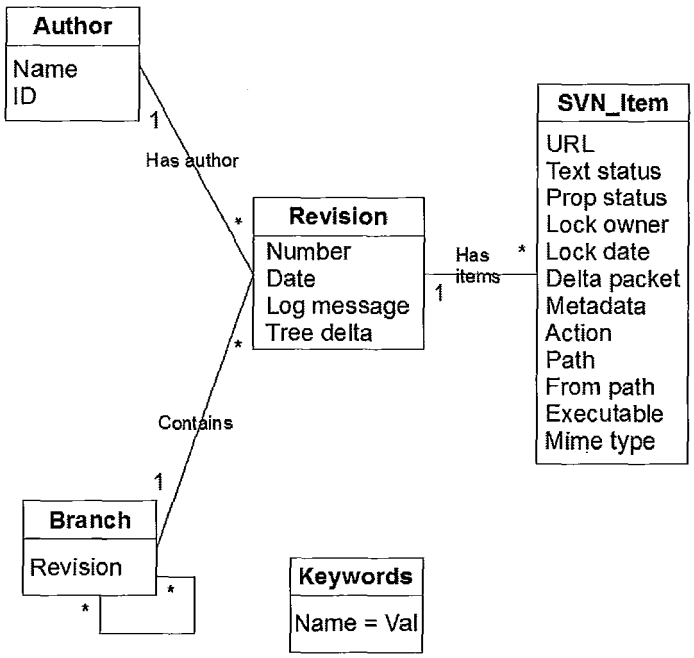

Figure 4.4: Revised UML model of SVN.

SVN_Item reflects SVN's versioning of both files and directories.

Note: Metadata is intentionally excluded from Figure 4.4.

Chapter 5

Models

5.1 Initial Models of SVN and CVS

MACHINE
SVN_01

SETS
FILESSET; *DATA*; *FILECONTENT*

OPERATIONS
Add1(*Files*) =
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\}$
THEN
skip
END;

As described earlier, the first model defines the sets, operations, argument types to operations and return types. Empty operation bodies are not allowed in Atelier B 4 so the *skip* command is inserted. It should be read as *nothing happens* in this context.

5.2 First Refinement of SVN and CVS

This second model details the client file system *ClientFiles*, VC repository *ServerRepository* and global version number *ServerVerNo* for SVN. CVS does not

have the global version number. All commands are split into client and server operations, like *Delete* and *CommitDelete*.

MACHINE
SVN_02

INCLUDES
SVN_01

VARIABLES
ClientFiles, *ServerRepository*, *ServerVerNo*

INVARIANT
$ClientFiles \in FILESSET \pfun FILECONTENT$
$\land\ ServerRepository \in (FILESSET \times \mathbb{N}) \pfun FILECONTENT$
$\land\ ServerVerNo \in \mathbb{N}$

$\land\ ((ServerRepository \neq \{\}) \iff (ServerVerNo > 0))$
$\land\ (\forall\ pp\ .\ (pp \in \mathbb{N} \land pp \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \Rightarrow pp \leq ServerVerNo))$

INITIALISATION
$ClientFiles := \{\} \parallel ServerRepository := \{\} \parallel ServerVerNo := 0$

Within the model *FILESSET* and *FILECONTENT* are the set of all items and item contents creatable within the file system.

Variables are defined in one section and typed in the next. *ClientFiles* is the set of items on the client computer. Initially it is empty. Defining *ClientFiles* as $FILESSET \pfun FILECONTENT$ does not limit us to a single directory system. Identifier *FILESSET* can later be refined to $(USER \times URL) \pfun FILECONTENT$ say where URL is a Universal Resource Locater. Carrying around these additional terms clutters the model unnecessarily.

Item histories are stored in the repository on the server, *ServerRepository*. It maps an item and a version number to contents at that version. It is also empty on initialization. *ServerVerNo* is the global version counter for the SVN repository. Initially it is zero.

5.2.1 Operations

Modeling Failure

Modeling VCS commands includes showing what happens when they fail or are called incorrectly. To do this the preconditions of an operation only type the input variables to the operation. All other checks are enforced as guards in an **IF** statement. An operation that creates files in the file system is an example:

$OSAddFile(NewFile) =$
PRE
$NewFile \in FILESSET \nrightarrow FILECONTENT \land NewFile \neq \{\}$
THEN
IF $\text{dom}(NewFile) \notin \text{dom}(ClientFiles)$ **THEN**
$ClientFiles := ClientFiles \cup \{NewFile\}$
ELSE
// Print helpful error message
END;
END;

For clarity this presentation does not use the if-else block. All guards are included in the precondition. Assume when an operation is called and the precondition fails, a "helpful error message" is output and the state space does not change.

Operations

Placing an item under version control is a two step process. First the client side operation *Add* adds the item to the list of items under version control on the client. Server operation *CommitAdd* uploads and stores the item to the VCS server for the first time.

These initial models are incomplete as there is no *Add* operation. No variable is defined on the client side to hold the items being added. To build a reasonably understandable model it is necessary to defer complexity to future extensions.

The following listings show the CommitAdd operation for CVS and then SVN:

$CommitAdd2(AFile) =$
PRE
$AFile \in FILESSET \land ClientFiles \neq \{\}$
$\land AFile \in \text{dom}(ClientFiles) \land AFile \notin \text{dom}(\text{dom}(ServerRepository))$

$\wedge\, (AFile \mapsto 1) \notin \mathrm{dom}(ServerRepository)$

THEN

$ServerRepository := ServerRepository$

$\cup\, \{(AFile \mapsto 1) \mapsto ClientFiles\,(AFile)\}$

END;

For SVN:

$CommitAdd2(Files) =$

PRE

$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$

$\wedge\, ServerVerNo + 1 \leq MAXINT \wedge Files \subseteq \mathrm{dom}(ClientFiles)$

$\wedge\, Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$

THEN

$ServerVerNo := ServerVerNo + 1 \parallel$

$ServerRepository := ServerRepository \cup$

$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN \in Files$

$\wedge\, Ver = ServerVerNo + 1 \wedge ServerVerNo + 1 \leq MAXINT$

$\wedge\, FileN \in \mathrm{dom}\ (ClientFiles) \wedge FileN \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge\, (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$

$|$

$ClientFiles\,(FileN))$

END;

Preconditions in the **PRE** block list the conditions to be satisfied for the operation to occur. Before an item can be placed under version control, it must exist, $AFile \in \mathrm{dom}(ClientFiles)$. It also must not already be under version control, $AFile \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$.

When preconditions for an operation are not met, state does not change. If the preconditions of all operations are not met the system is idle. When an operation is called and the preconditions are not met, a helpful error message describing what went wrong is displayed.

CommitAdd illustrates one of the differences between SVN and CVS: SVN supports transactions where CVS does not.

Transaction support is modeled by allowing multiple items in the SVN operation, $Files \subseteq FILESSET$. Within the body a lambda relation is

used to add items to the server repository. Informally the output is $(FileN,\ Ver) \nrightarrow ClientFiles(FileN)$. Formally the output is the same type as $ServerRepository$, $(FILESSET \times \mathbb{N}) \nrightarrow FILECONTENT$. $FileN \in Files$ examines each member of the input in turn and $(FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$ maintains the uniqueness of the domain of the $ServerRepository$ relation.

CVS operations only allows single items $AFile \in FILESSET$. A guard checks that $AFile$ is not in $ServerRepository$ already, turning the addition operation into a straight-forward union.

CommitAdd illustrates the difference between version numbering schemes. In SVN the transaction has a single version number shared by all items, $ServerVerNo + 1$. For CVS, each item has a version number that always begins at one, $(AFile \mapsto 1)$.

Observe $ServerVerNo$ is incremented each time it appears. This is necessary due to the use of $\|$, the parallel composition operator. Each statement is performed at the same time on different processors with no communication between them. As one cannot signal $ServerVerNo + 1$ to the others, each must perform it independently.

$Delete2(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
THEN
$ClientFiles := Files \lhd ClientFiles$
END;

SVN and CVS share the same delete operation. Preconditions assert the client item(s) exists, are under version control and removes them immediately. One small difference for CVS is single items must be turned into a set, $\{AFile\}$ to work with the domain restriction operator.

CVS does not support copy and move, so the next four operations apply to SVN only:

$Copy2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

THEN

$ClientFiles := ClientFiles \cup$

$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$

$\wedge\ NewToOldNames(NewF) \in \mathrm{dom}(ClientFiles)$

$\wedge\ NewF \notin \mathrm{dom}(ClientFiles)$

$|$

$ClientFiles(NewToOldNames(NewF)))$

END;

In *Copy* the mapping of new item names to existing items is a relation, $NewToOldNames \in FILESSET \nrightarrow FILESSET$. None of the new names must exist in the client file system, $\mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$ and all of the source names must exist there, $\mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$.

The lambda relation outputs $NewF \nrightarrow ClientFiles(NewToOldNames(NewF))$ of type $FILESSET \nrightarrow FILECONTENT$. That is, the new items created by the copy command. Statement $NewF \in \mathrm{dom}(NewToOldNames)$ iterates over all members of input *NewToOldNames* and $NewF \notin \mathrm{dom}(ClientFiles)$ maintains the uniqueness of the domain of *ClientFiles*.

$CommitCopy2(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$

$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerVerNo > 0$

$\wedge\ ServerVerNo + 1 \leq MAXINT$

$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$ // *Files now exist on client*

$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$

$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{ran}(NewToOldNames) = \{\}$

THEN

$ServerVerNo := ServerVerNo + 1 \parallel$

$ServerRepository := ServerRepository \cup$

$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N}$

$\wedge\ FileN \in \mathrm{dom}(NewToOldNames)$

$\wedge\ (NewToOldNames(FileN),\ Ver) \in \mathrm{dom}(ServerRepository)$

$\wedge\ (FileN,\ Ver) \notin \mathrm{dom}(ServerRepository)$

|

$ServerRepository(NewToOldNames(FileN),\ Ver))$

END;

Where *Copy2* affects only the client file system, *CommitCopy2* affects the server repository. It assumes the items have been copied on the client system already, $\mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$ and $\mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$, guaranteeing *Copy2* occurs before *CommitCopy2*.

The lambda relation in *CommitCopy2* is similar to the one in *Copy2*, though the output is of type $FILESSET \times \mathbb{N} \nrightarrow FILECONTENT$, the same type as *ServerRepository*.

Preconditions are the same as *Copy*, with some additions. The repository has to have a item to copy, $ServerRepository \neq \{\}$ and the size of the repository set has to be within the legal size of an integer $ServerVerNo + 1 \leq MAXINT$.

This lambda relation copies the full history of items as well. In $FileN \in \mathrm{dom}(NewToOldNames)$, *FileN* is the new item name. Within $(NewToOldNames(FileN),\ Ver) \in \mathrm{dom}(ServerRepository)$, $(NewToOldNames(FileN)$ is the existing item name, $(NewToOldNames(FileN),\ Ver)$ is the existing item name and any and all versions of it that exists in the repository. No restrictions have been placed on *Ver* so it ranges over all available values. If desired the lambda relation can be made more explicit by adding $Ver > 0 \wedge Ver \leq ServerVerNo$.

Associating the item history this way does not commit the model to copying and duplicating the history of an item. The association could be implemented with links to the source revisions, or a link to the last revision of the old item name.

Statement $ServerRepository \neq \{\}$ seems like a natural candidate to be an SVN invariant. It is rejected as operations *Add* and *CommitAdd* will always fail as they can never add the first item into the repository. Modifying it slightly gives a working invariant: $(ServerRepository \neq \{\}) \iff (ServerVerNo > 0)$

Guard $ServerVerNo + 1 \leq MAXINT$ appears almost universally throughout the model and seems a natural candidate for elevation to an invariant. If we do so AB 4 recursively generates a new PO, show $ServerVerNo + 1 + 1 \leq MAXINT$ in numerous places, leaving us no further ahead.

$Move2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
THEN
$ClientFiles := \mathrm{ran}(NewToOldNames) \lhd ((ClientFiles \cup$
$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$
$\wedge\ NewToOldNames(NewF) \in \mathrm{dom}(ClientFiles)$
$\wedge\ NewF \notin \mathrm{dom}(ClientFiles)$
$|$
$ClientFiles\,(NewToOldNames(NewF)))))$
END;

Operation *Move* is identical to Copy with one addition: The source items are deleted once the new items are created. $\mathrm{ran}(NewToOldNames) \lhd ((ClientFiles$ ... uses domain restriction to eliminate source items from the *ClientFiles* relation.

$CommitMove2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{ran}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ServerVerNo := ServerVerNo + 1 \parallel$

$ServerRepository := ServerRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N}$
$\wedge\ FileN \in \mathrm{dom}(NewToOldNames) \wedge Ver \leq ServerVerNo$
$\wedge\ ServerVerNo + 1 \leq MAXINT$
$\wedge\ NewToOldNames(FileN) \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge\ (NewToOldNames(FileN),\ Ver) \in \mathrm{dom}(ServerRepository)$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
|
$ServerRepository(NewToOldNames(FileN),\ Ver))$

END;

Guards for *CommitMove* are constructed to insure *Move* occurs first. New names exist in client items and does not exist in the server repository where old names do not exist in the client items and do exist in the server repository.

$Committed2 \longleftarrow Commit2(AFile) =$
PRE
$Committed2 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ServerRepository := ServerRepository \cup$
$\lambda\ (FileN,\ Ver \bullet FileN \in FILESSET$
$\wedge\ Ver \in \mathbb{N} \wedge FileN = AFile$
$\wedge\ Ver = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1$
$\wedge\ FileN \in \mathrm{dom}(ClientFiles) \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
|
$ClientFiles\,(FileN))\ \|$

$Committed2 := AFile$
END;

Commit2 for CVS is notable as the first operation containing a return value, *Committed2* as well as a nested set comprehension.

As the two statements are performed in parallel, *Committed2* always returns the name of the item committed, *AFile* regardless of the success or failure of the lambda relation. If the lambda relation fails, should not *Committed2* be the empty set, {}?

The guards and conditions of the lambda relation are repeated in the precondition block. Preconditions are expressed in terms of *AFile*, lambda relations in terms of

FileN. Statement $FileN = AFile$ in the lambda relation ensures the same guards apply.

Set comprehension $\{ww \mid ww \in \mathbb{N} \wedge ...$ is tasked with finding the largest version number of *AFile* already in the repository, add one to it and return it to the parent lambda relation through the variable *Ver*. It is the version number of the latest revision of *AFile*:

$$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT \\ \wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1$$

It reappears in many places throughout the models and is worth noting and understanding.

$$Updated2 \longleftarrow Update2(Files) =$$

PRE

$$Updated2 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\} \\ \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \\ \wedge Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$$

THEN

$$ClientFiles := (Files \lhd ClientFiles) \cup \\ \lambda\ FileN.(FileN \in FILESSET \\ \wedge FileN \in Files \wedge FileN \notin \mathrm{dom}(ClientFiles) \\ \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge FileN \notin \mathrm{dom}(ClientFiles) \\ | \\ ServerRepository(FileN \mapsto \max(\\ \{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT \\ \wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1)) \parallel$$

// Merge if changes are disjoint

// User intervention required if changes overlap

$$Updated2 := Files$$

END;

Update2 from SVN merges any changes made by others into local items a user has checked out. There are three cases to consider: First, the item has not changed on the server. Second, there are changes but they do not overlap with local changes. In the third the changes overlap.

Merging and reconciling overlapping edits are described as comments. File con-

tents is defined as being a member of the set *FILECONTENTS*. It is intentionally not defined well enough to have contents to compare so the functionality cannot be implemented at this level of abstraction. What the model does, replacing the client item with the server item is of course incorrect. It is a place-holder for a properly implemented future Refinement.

As the major focus was on modeling the most commonly used aspects of SVN, merging and reconciliation are not pursued further. It is one open area for potential future work.

$$
\begin{array}{l}
\mathit{CheckedOut2} \longleftarrow \mathit{CheckOut2}(\mathit{Files}) = \\
\textbf{PRE} \\
\quad \mathit{CheckedOut2} \subseteq \mathit{FILESSET} \wedge \mathit{Files} \subseteq \mathit{FILESSET} \wedge \mathit{Files} \neq \{\} \\
\quad \wedge\, \mathit{ServerRepository} \neq \{\} \wedge \mathit{Files} \subseteq \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository})) \\
\quad \wedge\, \mathit{Files} \cap \mathrm{dom}(\mathit{ClientFiles}) = \{\} \\
\textbf{THEN} \\
\quad \mathit{ClientFiles} := (\mathit{Files} \lhd \mathit{ClientFiles}) \cup \\
\quad\quad \lambda\ \mathit{FileN}.(\mathit{FileN} \in \mathit{FILESSET} \wedge \mathit{FileN} \in \mathit{Files} \\
\quad\quad\quad \wedge\, \mathit{FileN} \in \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository})) \\
\quad\quad\quad \wedge\, \mathit{FileN} \notin \mathrm{dom}(\mathit{ClientFiles}) \\
\quad\quad\quad | \\
\quad\quad\quad \mathit{ServerRepository}(\mathit{FileN} \mapsto \max(\\
\quad\quad\quad\quad \{ww \mid ww \in \mathbb{N} \wedge ww \leq \mathit{MAXINT} \\
\quad\quad\quad\quad \wedge\, (\mathit{FileN} \mapsto ww) \in \mathrm{dom}(\mathit{ServerRepository})\}) + 1)) \parallel \\
\\
\quad \mathit{CheckedOut2} := \mathit{Files} \\
\textbf{END};
\end{array}
$$

To check-out items in SVN, they must already be in the repository. Precondition $\mathit{Files} \cap \mathrm{dom}(\mathit{ClientFiles}) = \{\}$ specifies items must not exist in the client item system. It is an optional guard who gives more safety to the client: If the item already exists locally it will not be overwritten. The lambda relation demonstrates how to overwrite said items using the domain restriction operator to remove items with the same name. Both statements together are contradictory. One or the other should be used.

In CVS *CheckOut* is similar, though restricted to searching for a a single item, *AFile*, in the lambda relation through the $\mathit{FileN} = \mathit{AFile}$ clause.

$Reverted2 \longleftarrow Revert2(FilesToRevertVer) =$
PRE
$Reverted2 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ ServerVerNo + 1 \leq MAXINT$
THEN
$ClientFiles := (\mathrm{dom}(FilesToRevertVer) \lhd ClientFiles) \cup$
$\lambda\ FileNm.(FileNm \in FILESSET \wedge FileNm \in \mathrm{dom}(FilesToRevertVer)$
$\wedge\ (FileNm \mapsto FilesToRevertVer(FileNm)) \in \mathrm{dom}(ServerRepository)$
$\wedge\ FileNm \notin \mathrm{dom}(ClientFiles)$
$|$
$ServerRepository(FileNm \mapsto FilesToRevertVer(FileNm))) \parallel$

$Reverted2 := \mathrm{dom}(FilesToRevertVer)$
END

Revert2 from SVN accepts as an argument the relation *FilesToRevertVer* of type $FILESSET \nrightarrow \mathbb{N}1$ where *FILESSET* is the item name to revert and N is the version to revert to. There is no precondition to check if the revert-to versions exist. Instead it is a guard within the lambda relation:

$(FileNm \mapsto FilesToRevertVer(FileNm)) \in \mathrm{dom}(ServerRepository)$

This allows each entry within the *FilesToRevertVer* relation to succeed or fail individually.

5.3 Second Refinement: Client Version Number

SVN_03 includes and extends machine SVN_02 described previously. This machine adds tracking the version number of checked out items through variable *ClientVersionNo*.

Operations *CommitAdd*, *Copy* and *Commit* amongst others add or update members of *ClientVersionNo*. *CommitDelete* removes members while *CommitMove* does both. Representative operations *CommitAdd* and *CommitDelete* are detailed below.

Note that as in the previous section, *FILESSET* can later be refined to $(USER \times URL) \nrightarrow FILENAME$ say to extend the model to cover multiple users and locations.

MACHINE
SVN_03

INCLUDES
SVN_02

VARIABLES
ClientVersionNo

INVARIANT
$ClientVersionNo \in FILESSET \nrightarrow \mathbb{N}$ // *Can shrink*

INITIALISATION
$ClientVersionNo := \{\}$

$CommitAdd3(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
THEN
$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files$
$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$ServerVerNo + 1) \parallel$

$CommitAdd2(Files)$
END;

Operations that refine existing operations of the same name must at a mini-

mum contain the precondition of the parent operation. It can be more specialized as *CommitAdd3* for SVN is, adding 2 additional clauses over *CommitAdd2*. Refined operations may read variables from parent machines but cannot write to them. Only the machine declaring the variable may write to it. Writing to these variables is accomplished by calling the operation in the machine. This is done to remove dependencies between machines.

Like the wine dark sea of the Iliad, the body of *CommitAdd3* is a recurring definition reappearing throughout the models: Increment server counter (for SVN), lambda relation and call to parent function where it exists. Small changes lead to the CVS implementation: $Files \subseteq FILESSET$ to $AFile \in FILESSET$ and lambda relation condition from $FileN \in Files$ to $FileN = AFile$.

CommitDelete3(*Files*) =
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
THEN
$ClientVersionNo := Files \lhd ClientVersionNo$
END;

The above listing is *CommitDelete3* for SVN. CVS's implementation is derived by changing precondition $Files \subseteq FILESSET$ to $AFile \in FILESSET$ and range subtraction from *Files* to $\{AFile\}$. CVS has no transaction support.

5.4 Third Refinement: Under Version Control List for the Server

MACHINE
SVN_04

INCLUDES
SVN_03

VARIABLES
ServerUVC

INVARIANT
$ServerUVC \subseteq FILESSET$

// *ServerUVC Invariants*

$\wedge\ (ServerUVC \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)))$

INITIALISATION
$ServerUVC := \{\}$

SVN_04 keeps track of items under version control through variable *ServerUVC* is added here. It must be separate from *ServerRepository* as once an item is added to the repository it is never touched again. Items cannot be removed from the repository when the VC system is instructed to no longer version it. A list of items under version control must necessarily be different from a list of items checked out as an item can be under version control but not checked out by any clients.

$CommitMove4(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerUVC \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

$\wedge\, \mathrm{ran}(\mathit{NewToOldNames}) \cap \mathrm{dom}(\mathit{ClientFiles}) = \{\}$
$\wedge\, \mathrm{dom}(\mathit{NewToOldNames}) \cap \mathrm{dom}(\mathit{ClientVersionNo}) = \{\}$
$\wedge\, \mathrm{ran}(\mathit{NewToOldNames}) \subseteq \mathrm{dom}(\mathit{ClientVersionNo})$
$\wedge\, \mathrm{dom}(\mathit{NewToOldNames}) \cap \mathit{ServerUVC} = \{\}$
$\wedge\, \mathrm{ran}(\mathit{NewToOldNames}) \subseteq \mathit{ServerUVC}$
$\wedge\, \mathrm{dom}(\mathit{NewToOldNames}) \cap \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository})) = \{\}$
$\wedge\, \mathrm{ran}(\mathit{NewToOldNames}) \subseteq \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository}))$

THEN

$\mathit{ServerUVC} := (\mathit{ServerUVC} \cup \mathrm{dom}(\mathit{NewToOldNames})) - \mathrm{ran}(\mathit{NewToOldNames}) \parallel$

CommitMove3(*NewToOldNames*)

END;

CommitMove4 for SVN neatly encapsulates the use of *ServerUVC*: *Move* has already been called, so the old names in *ClientFiles* have been deleted and the new names added. Observe that the preconditions involving *ClientFiles* checks for this. As the move operation hasn't been performed on the server yet, old names exist there and new names do not. New items are added to the server $\mathit{ServerUVC} \cup \mathrm{dom}(\mathit{NewToOldNames})$ and the old ones removed $- \mathrm{ran}(\mathit{NewToOldNames})$.

5.5 Fourth Refinement: Shadow Under Version Control List for the Client

VARIABLES

ShadowUVC

INVARIANT

$\mathit{ShadowUVC} \subseteq \mathit{FILESSET}$

INITIALISATION

$\mathit{ShadowUVC} := \{\}$

SVN_05 adds a shadow list of items under version control on the client side. It exists to let the CVS and SVN clients record an arbitrary number of add and delete operations on an item before it is committed. Operation *Move5* is the same as

CommitMove4 from the previous model. It is presented without further comment.

Move5(NewToOldNames) =
PRE
NewToOldNames $\in$ *FILESSET* $\nrightarrow$ *FILESSET* $\wedge$ *NewToOldNames* $\neq \{\}$
$\wedge$ *ClientFiles* $\neq \{\}$ $\wedge$ *ShadowUVC* $\neq \{\}$
$\wedge$ dom(*NewToOldNames*) $\cap$ dom(*ClientFiles*) $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ dom(*ClientFiles*)
$\wedge$ dom(*NewToOldNames*) $\cap$ *ShadowUVC* $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ *ShadowUVC*
THEN
Move4(NewToOldNames) ||

ShadowUVC := (*ShadowUVC* $\cup$ dom(*NewToOldNames*)) $-$ ran(*NewToOldNames*)
END;

5.6 Fifth Refinement: Shadow Repository and Version Number for the Client

Similar to the first model, SVN_06 adds a repository and version number for SVN only to the client side. As in the previous model they are shadows who add support to the client for an arbitrary number of add, delete, copy and move operations on items before commit is called. Editing an item is supported as every add operation assigns a new shadow version number to the item.

VARIABLES
ShadowRepository, *ShadowVerNo*

INVARIANT
ShadowRepository $\in$ (*FILESSET* $\times \mathbb{N}$) $\nrightarrow$ *FILECONTENT*
$\wedge$ *ShadowVerNo* $\in \mathbb{N}$

// ------ *Invariants* ------

$\wedge$ ((*ShadowRepository* $\neq \{\}$) $\iff$ (*ShadowVerNo* > 0))

$\wedge\ (\forall\ pp\ .\ (pp\ \in \mathbb{N} \wedge pp\ \in \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\Rightarrow pp\ \leq ShadowVerNo))$

$\wedge\ (ShadowUVC\ \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)))$

INITIALISATION

$ShadowRepository\ := \{\}\ \|ShadowVerNo\ := 0$

Operation *Update6* for SVN illustrates the use of the shadow repository. Preconditions checks that items checked out exist in the server repository, are under version control, are checked out by the client, are in the shadowed under version control list, are in the shadow repository and are in the client items list.

The lambda relation contains the same comprehension set nested within it twice:

$\{uu \mid uu\ \in \mathrm{ran}(\mathrm{dom}(ServerRepository))\ \wedge uu\ \leq ServerVerNo$
$\wedge\ (FileN\ \mapsto uu)\ \in \mathrm{dom}(ServerRepository)\}$

It determines the largest version number of item *FileN* in the server repository. That is, the version number of the most recently checked in copy of *FileN*. If the contents are not the same as the item from the client item $ServerRepository(FileN \mapsto \mathrm{max...}) \neq ClientFiles(FileN)$, said client item needs updating in the shadow repository.

Note that for the SVN model, there is no relationship between the shadow version counter, *ShadowVerNo* and the server version counter *ServerVerNo*. They vary independently as client and server operations can be called in arbitrary orders.

$Updated6 \longleftarrow Update6(Files) =$

PRE

$Updated6 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\}\ \wedge ShadowUVC \neq \{\}\ \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\}\ \wedge ServerUVC \neq \{\}\ \wedge ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0\ \wedge ShadowVerNo + 1\ \leq MAXINT$
$\wedge\ ServerVerNo > 0\ \wedge ServerVerNo + 1\ \leq MAXINT$
$\wedge\ Files\ \subseteq \mathrm{dom}(ClientFiles)\ \wedge Files\ \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files\ \subseteq ShadowUVC\ \wedge Files\ \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files\ \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge Files\ \subseteq ServerUVC$
$\wedge\ ShadowVerNo + 1\ \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$

THEN

$ShadowVerNo := ShadowVerNo + 1 \parallel$

$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N}$
$\wedge\ Ver = ShadowVerNo + 1 \wedge FileN \in Files$
$\wedge\ ServerRepository(FileN \mapsto \max($
$\{uu \mid uu \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge uu \leq ServerVerNo$
$\wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\}))$
$\neq ClientFiles(FileN)$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
$|$
$ServerRepository(FileN \mapsto \max($
$\{uu \mid uu \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge uu \leq ServerVerNo$
$\wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\}))) \parallel$

$Updated6 \longleftarrow Update5(Files)$

END;

5.7 Sixth Refinement: Status Operation

SETS

$STATUS = \{Added, Deleted, NoChange, MostRecent, OutOfDate\}$

...

OPERATIONS

$Statuses \longleftarrow Status7(Files) =$

PRE

$Statuses \in FILESSET \nrightarrow STATUS \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$

THEN

// Single statement begins here

$Statuses := (((($

// 1. Added

λ *File1*.(*File1* $\in$ *FILESSET* $\wedge$ *File1* $\in$ *Files*
$\wedge$ *File1* $\in$ dom(*ClientFiles*) $\wedge$ *File1* $\in$ *ShadowUVC*
$\wedge$ *File1* $\in$ dom(dom(*ShadowRepository*)) $\wedge$ *File1* $\notin$ dom(*ClientVersionNo*)
$\wedge$ *File1* $\notin$ *ServerUVC* $\wedge$ *File1* $\notin$ dom(dom(*ServerRepository*))
$\wedge$ *File1* $\notin$ dom(*Statuses*)
|
Added)) $\cup$
...

END

SVN_07 adds a *status* operation. It returns the status of each item with respect to the repository. Many calls to CVS and SVN functions return the status of items as part of their output, thus the decision to write status as an operation with the output as a return variable instead of a persistent global variable.

An item is in the *Added* state when it is in the client relations *ClientFiles*, *ShadowUVC* and *ShadowRepository* and not in the server relations *ClientVersionNo*, *ServerUVC* and *ServerRepository*. It has been newly added to the client and is waiting for a submit operation to the server.

Items are in the *Deleted* state when they have been deleted from the client relations *ClientFiles* and *ShadowUVC*. They are still in *ShadowRepository* as these items have been previously checked-in. In effect they are newly deleted on the client side and are waiting for a submit operation to delete them from the server relations *ClientVersionNo* and *ServerUVC*. Relation *ServerRepository* is of course not affected.

When items are under version control and are actively being edited and checked-in by multiple users we want to know what the status of them is with respect to the server copies.

Status *NoChange* means the client item is up-to date with respect to the server repository item. This is checked by comparing the contents of the client item with the most recent version of the item in the server repository:

ClientFiles(*File5*) = *ServerRepository*(*File5* $\mapsto$ max(
{*v5* | *v5* $\in$ ran(dom(*ServerRepository*)) $\wedge$ *v5* $\leq$ *ServerVerNo*
$\wedge$ (*File5* $\mapsto$ *v5*) $\in$ dom(*ServerRepository*) $\wedge$ *ServerRepository* $\neq$ {}}))

When changes have been made to a checked out item that have not been committed yet the status is set to *MostRecent*. The lambda relation is the same as the no change case with the equality of the nested lambda relation inverted,

$ClientFiles(File6) \neq ServerRepository(....$

Lastly an item is out of date with respect to the server *OutOfDate* when the version number in *ServerVersionNo* is less than the maximum version number of that item in *ServerRepository*. User B has committed a change to the item after user A checked it out, so A is out of date.

This set comprehension shows the out of date check for Status7 for SVN:

$$ClientVersionNo(File7) < \max(\{v7 \mid v7 \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge v7 \leq ServerVerNo \wedge (File7 \mapsto v7) \in \mathrm{dom}(ServerRepository) \wedge ServerRepository \neq \{\}\})$$

5.8 Seventh Refinement: Binary File Support

VARIABLES

ServerBinaryFile, *ServerCOB*

INVARIANT

$ServerBinaryFile \subseteq FILESSET \wedge$
$ServerCOB \subseteq FILESSET$

INITIALISATION

$ServerBinaryFile := \{\} \,||\, ServerCOB := \{\}$

Binary file support is automatically handled by SVN. In CVS users are expected to remember which files are binary and mark them with command line parameters.

Binary items are identified by their membership in the *IsBinaryFile* set. Who has checked out a particular binary item is recorded in the *RepositoryCheckedOutBinary* set.

$Add8(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\, Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\, Files \cap ServerUVC = \{\} \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\, Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\, ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\, ShadowVerNo \geq 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\, ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
THEN
// SVN uses binary detection here. We simulate it by making a random
// subset of input set Files binary.
ANY *IsBinary*
WHERE $IsBinary \subseteq Files$
THEN
$ServerBinaryFile := ServerBinaryFile \cup$
$\{FileN \mid FileN \in FILESSET \wedge FileN \in Files \wedge FileN \in IsBinary$
$\wedge\, FileN \notin ServerBinaryFile\} \,||$

Add7(Files)
END
END;

SVN_08 in SVN describes adding binary files on the client side. When items are added to SVN the **ANY...WHERE** statements simulates adding binary items by creating a local set *IsBinary* that is a random subset of *Files*. All of the members are binary items who are added to the *IsBinaryFile* set by the lambda relation.

Binary items should never need updating. For an update to occur, another user must have checked out the item - which is impossible since it is already locked. Even though the model does not explicitly support multiple users, *Update* is written to remove all binary items from the set of items to be updated before update is called. For SVN it is:

Updated8 $\longleftarrow$ *Update8(Files)* =
PRE
...
// *Once binary files are removed from the update set, it isn't empty*
// $\wedge$ *Files* $-$ $\{FileB \mid FileB \in Files \wedge FileB \in ServerCOB\} \neq \{\}$
THEN
// *Updates don't occur for binary files*
LET *NotBinary* **BE**
NotBinary = *Files* $-$ $\{FileB \mid FileB \in Files \wedge FileB \in ServerCOB\}$
IN
IF *NotBinary* $\neq \{\}$ **THEN**
Updated8 $\longleftarrow$ *Update7(NotBinary)*
END
END

END;

The precondition asserts that there are non-binary items in the *Files* set. This is necessary as the **LET** statement in the body has to type the *NotBinary* local variable. If *NotBinary* evaluates to the empty set it has no type. Given this, **IF** *NotBinary* $\neq \{\}$ **THEN** ... is still required to convince the prover *NotBinary* is indeed not the empty set.

CVS does not automatically track binary items. It is up to users to remember and specify all binary items when interacting with the repository. A representative operation is *Add*:

Add8(*AFile*, *SpecBinary*) =
PRE
AFile $\in$ *FILESSET* $\wedge$ *SpecBinary* $\in$ *BOOL* $\wedge$ *ClientFiles* $\neq \{\}$
$\wedge$ *AFile* $\in$ dom(*ClientFiles*) $\wedge \{AFile\} \cap$ dom(dom(*ServerRepository*)) $= \{\}$
$\wedge \{AFile\} \cap$ *ServerUVC* $= \{\} \wedge \{AFile\} \cap$ dom(*ClientVersionNo*) $= \{\}$
$\wedge \{AFile\} \cap$ dom(dom(*ShadowRepository*)) $= \{\}$
$\wedge \{AFile\} \cap$ *ShadowUVC* $= \{\} \wedge$ *AFile* $\notin$ *ServerBinaryFile*
THEN
ANY *IsBinary*
WHERE *IsBinary* $\in$ *BOOL*
THEN
// *User identifies item incorrectly*
IF (*IsBinary* $=$ *TRUE* $\wedge$ *SpecBinary* $=$ *FALSE*)
$\vee$ (*IsBinary* $=$ *FALSE* $\wedge$ *SpecBinary* $=$ *TRUE*) **THEN**
skip
ELSE
IF *IsBinary* $=$ *TRUE* $\wedge$ *SpecBinary* $=$ *TRUE* **THEN**
ServerBinaryFile $:=$ *ServerBinaryFile* $\cup \{AFile\}$
END ||
Add7(*AFile*)
END
END
END;

If an item is incorrectly identified as binary or text it will be mangled by CVS. This is represented by not passing item *AFile* to operation *Add7* in the sixth model. For an incorrect specification, *skip* is called, indicating an error has occurred.

The use of **ANY** ... **WHERE** may suggest specifying an item correctly is a matter of luck. This is of course not the case. The operation is written this way to be consistent with the SVN model. An alternative is to remove **ANY** ... **WHERE** and

turn *IsBinary* into an input of type BOOL to the operation.

5.9 Eighth Refinement: Pristine Cache

This model introduces the client side cache, or *pristine cache* on the client system for SVN. CVS does not have this feature. It stores items in the state they were in when checked-out, updated or reverted. User changes to items are never stored to the pristine cache. Pristine items are only updated when they are pulled from the server. Having pristine items stored locally gives the SVN user options to revert and diff with respect to the pristine item. The server does not need to be involved. All of these cached items are automatically managed by SVN in hidden sub-directories that users should not touch.

VARIABLES

$ClientPristine$

INVARIANT

$ClientPristine \in FILESSET \times \mathbb{N} \nrightarrow FILECONTENT$

INITIALISATION

$ClientPristine := \{\}$

ClientPristine is a relation of the same type as *ServerRepository*.

$CheckedOut9 \longleftarrow CheckOut9(Files) =$

PRE

...

THEN

$ClientPristine := ClientPristine \cup$

$\lambda\ FileP,\ VerP.(FileP \in FILESSET \wedge VerP \in \mathbb{N} \wedge FileP \in Files$

$\wedge\ FileP \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$

$\wedge\ FileP \in ServerUVC \wedge FileP \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge\ FileP \mapsto VerP \in \mathrm{dom}(ServerRepository)$

$\wedge\ FileP \mapsto VerP \notin \mathrm{dom}(ClientPristine)$

$\wedge\ VerP = \max($

$\{vP \mid vP \in \mathbb{N} \wedge vP \leq ServerVerNo$

$\wedge\ (FileP \mapsto vP) \in \mathrm{dom}(ServerRepository)\})$

$|$

$ServerRepository(FileP,\ VerP))$ ||

$CheckedOut9 \longleftarrow CheckOut8(Files)$

END;

As in previous cases the nested lambda relation returns the maximum version number of *FileP* in *ServerRepository*.

5.10 Invariants

Invariants constrain the state space and transformations of the model. These restrictions describe how the state space evolves over time and help the prover assert it works correctly. In an operation the preconditions are initially examined to see if it is eligible for execution. Invariants are checked as well as they must always hold. As a variable is only manipulated in one statement of the operation body due to parallel execution, invariants involving it are asserted again after the statement.

The client-server nature of the model and the two steps involved in Add, Delete, Move and Copy make finding invariants between client variables (*ClientFiles, ShadowUVC, ShadowRepository* and *ClientVersionNo*) and server variables (*ServerRepository, ServerVersionNo, ServerUVC, ServerCOB* and *ServerIsBinary*) difficult. Many attempted invariants are also invalidated because the sets and relations shrink as well as grow.

In the next two sections we look at and characterize successful and unsuccessful invariants.

5.10.1 Unsuccessful Candidates

Unsuccessful invariants give insights into what can and cannot be done with the models. Two are described here:

Hypothesis 1: All items checked out by the client are recorded in the SVN server:

$ShadowUVC \subseteq \text{dom}(\text{dom}(ServerRepository))$

Counter-example: From *Copy5* and *Move5* in Refinement four we have:

$ShadowUVC := ShadowUVC \cup \text{dom}(NewToOldNames)$

Derivation:

$\forall\, x.(x \in ShadowUVC \cup \mathrm{dom}(NewToOldNames) \Rightarrow x \in \mathrm{dom}(\mathrm{dom}(ServerRepository)))$

$x \in ShadowUVC \cup \mathrm{dom}(NewToOldNames) \Rightarrow x \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$True \Rightarrow \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$True \Rightarrow False$

$False$

Operations *Copy5* and *Move5* affect only client relation *ShadowUVC*. Server operations *CommitCopy4* and *CommitMove4* must be invoked by a check-in event to modify *ServerRepository*.

If copying and moving items was an immediate operation this invariant would hold. As it is performed in two steps, it suggests investigating invariants that hold on the client side only and server side only.

Hypothesis 2: For the model of SVN, *ClientPristine* relates to other server-side variables as follows:

$\mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\, \mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq ServerUVC$
$\wedge\, \mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\, ServerBinaryFile \subseteq \mathrm{dom}(\mathrm{dom}(ClientPristine))$
$\wedge\, ServerCOB \subseteq \mathrm{dom}(\mathrm{dom}(ClientPristine))$

Here the prover isn't able to prove any of the hypothesis during it's default alloted time of two minutes. They are all believed to be true, but are commented out in the model to represent their tenousness.

5.10.2 Successful Candidates

Using the previous section as a guide, a number of successful invariants were implemented for the server. They are listed here without proof. The best we can say is, Atelier B 4 did not find any counter-examples for them:

Hypothesis 1: From SVN_02, when items are under version control, the version number counter is greater than zero:

$ServerRepository \neq \{\} \iff ServerVerNo > 0$

Hypothesis 2: When items are under version control, the largest version number in the repository is equal to the version number counter:

$$ServerVerNo > 0 \Rightarrow (ServerVerNo = \max(\mathrm{ran}(\mathrm{dom}(ServerRepository))))$$

It is necessarily long winded to include, *when items are under version control* as, when there are no items under version control, there is no largest version number in the empty repository:

$$ServerVerNo = 0 \Rightarrow (ServerVerNo = \max(\mathrm{ran}(\mathrm{dom}(ServerRepository))))$$
$$True \Rightarrow (0 = \max(\{\}))$$
$$False$$

Hypothesis 3: From SVN_02, all version numbers in the server repository are less than or equal to the repository version number counter:

$$\forall\, xx \;.\; (xx \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \Rightarrow xx \leq ServerVerNo)$$

Hypothesis 4: All version numbers from one to the repository version number counter inclusive are in the server repository:

$$\forall\, xx \;.\; (xx \in \mathbb{N} \wedge xx \leq ServerVerNo \Rightarrow xx \in \mathrm{ran}(\mathrm{dom}(ServerRepository)))$$

These four invariants assert relationships between only two variables. That they only increase and increase in lock step with each-other shows they are tightly coupled. Analogously similar invariants can be made for *ShadowRepository* and *ShadowVersionNo*.

5.11 Proof Obligations

Proof obligations generated by Atelier B 4 come in three forms: Showing the initialization of sets and relations obeys their type rules, showing that adding and removing members from sets and relations obeys their type rules and showing invariants are always obeyed. We give an example of each.

1. Example initialization proof:

From SVN_02, check that invariant:

$$ClientFiles \in FILESSET \nrightarrow FILECONTENT$$

is established by the initialization:

$$\{\} \in FILESSET \nrightarrow FILECONTENT$$
$$True$$

2. Example add/remove proof:
From Move2 in SVN_02, check that invariant:

$$ClientFiles \in FILESSET \pfun FILECONTENT$$

is preserved by the operation:

$$\begin{array}{l}\mathrm{ran}(NewToOldNames) \domrestrict ((ClientFiles \cup \lambda NewF.(\\ \quad NewF \in FILESSET \land NewF \in \mathrm{dom}(NewToOldNames) \\ \quad \land NewToOldNames(NewF) \in \mathrm{dom}(ClientFiles) \land NewF \notin \mathrm{dom}(ClientFiles) \\ \quad | \\ \quad ClientFiles(NewToOldNames(NewF))))) \in FILESSET \pfun FILECONTENT\end{array}$$

From the guards:

$$NewToOldNames \in FILESSET \pfun FILESSET$$

Therefore $\mathrm{dom}(NewToOldNames) \in FILESSET$
Within the lambda relation

$$\begin{array}{l}NewF \in FILESSET \\ ClientFiles(\mathrm{ran}(NewToOldNames)) \in FILECONTENT\end{array}$$

Therefore the lambda relation evaluates to $FILESSET \pfun FILECONTENT$
Perform the type substitutions:

$$\begin{array}{l}\mathrm{ran}(FILESSET \pfun FILESSET) \domrestrict ((FILESSET \pfun FILECONTENT \\ \quad \cup FILESSET \mapsto FILECONTENT)) \in FILESSET \pfun FILECONTENT \\ FILESSET \domrestrict (FILESSET \pfun FILECONTENT) \\ \quad \in FILESSET \pfun FILECONTENT \\ FILESSET \pfun FILECONTENT \in FILESSET \pfun FILECONTENT \\ True\end{array}$$

This is the most complicated example in that it first adds the lambda relation due to the bracketing, then removes $\mathrm{ran}(NewToOldNames)$.

3. Example invariant preservation proof:
In operation Commit2 from SVN_02, given:

$$\begin{array}{l}1 \leq ServerVerNo + 1 \land \\ xx \in \mathrm{ran}(\mathrm{dom}(ServerRepository \cup \{AFile \mapsto ServerVerNo + 1 \\ \quad \mapsto AFileContents\}))\end{array}$$

Check that invariant:

$(\forall\ pp.(pp \in \text{ran}(\text{dom}(ServerRepository)) \Rightarrow pp \leq ServerVerNo))$

is preserved by the operation:

$xx \leq ServerVerNo + 1$

Intuitively, show that all version numbers in the repository are less than or equal to the repository version number counter *ServerVerNo*. The invariant specifically checks when a new check-in is performed, the new items added with version number $ServerVerNo + 1$ are less than or equal to the new repository counter value, $ServerVerNo + 1$.

We know this is true because a check-in is only accepted when one or more items has changes in it, or on a add, copy, move or delete. For delete, a item is removed from the version control set and the repository is unchanged. For all others, a new directory tree is made on the server for the operation with version number $ServerVerNo + 1$. At the same time, *ServerVerNo* is incremented by one. By induction on this coupling we show the invariant holds:

From the hypothesis:

$$ServerVerNo \geq 0 \wedge ServerVerNo + 1 \geq 1 \wedge$$
$$xx \in \text{ran}(\text{dom}(ServerRepository)) \wedge xx \in (\text{ran}(\text{dom}(ServerRepository)) \cup (ServerVerNo + 1))$$

As the guard on the invariant has ServerVerNo > 0, we strengthen our derived hypothesis:

$$ServerVerNo > 0 \wedge ServerVerNo + 1 > 1 \wedge$$
$$xx \in \text{ran}(\text{dom}(ServerRepository)) \wedge xx \in (\text{ran}(\text{dom}(ServerRepository)) \cup (ServerVerNo + 1))$$

This is the basis of an inductive proof:

Base case: 1. $ServerVerNo = 1 \Rightarrow 1 \in \text{ran}(\text{dom}(ServerRepository))$

Inductive case: 2. $ServerVerNo \mathrel{+}= 1 \Rightarrow RepositoryVersion \mathrel{+}= 1 \in (\text{ran}(\text{dom}(ServerRepository)) \cup (ServerVerNo + 1))$

Rewritten:

$1 \Rightarrow 1 \in \{1\}$ *from* 1.
$1 + 1 \Rightarrow 1 + 1 \in (\{1\} \cup \{2\})$ *from* 2.
$2 \Rightarrow 2 \in \{1, 2\}$
$2 + 1 \Rightarrow 2 + 1 \in (\{1, 2\} \cup \{3\})$ *from* 2.

$3 \Rightarrow 3 \in \{1, 2, 3\}$

...

Returning to the invariant to prove:

$\forall xx.(xx \in \text{ran}(\text{dom}(ServerRepository)) \Rightarrow xx \leq ServerVerNo$

By induction:

$\forall xx.(xx \in \{1\} \Rightarrow xx \leq 1)$
$\forall xx.(xx \in \{1, 2\} \Rightarrow xx \leq 2)$
$\forall xx.(xx \in \{1, 2, 3\} \Rightarrow xx \leq 3)$

...

True

By induction the invariant is preserved.

5.12 Lemma Proof Obligations

Lemma proofs are helper proofs derived by the prover to help simply more complicated proofs. They are stored separately from the regular proof obligations. We look at two here, one provable and one not:

Lemma 1: For the *Commit3* operation in file SVN_03 Atelier B must prove the lambda relation generating the item history never returns an empty set as a result.

Hypothesis:

$$btrue \Rightarrow \neg(\{uu \mid uu \in INTEGER \wedge 0 \leq uu \wedge uu \leq MAXINT \wedge uu \leq ServerVerNo \wedge AFile \mapsto uu \in \text{dom}(ServerRepository)\} = \{\})$$

$$\neg(\{uu \mid uu \in INTEGER \wedge 0 \leq uu \wedge uu \leq MAXINT \wedge uu \leq ServerVerNo \wedge AFile \mapsto uu \in \text{dom}(ServerRepository)\} = \{\}) \ (By\ MP)$$

Atelier B defines $\mathbb{N}$ as:

$$uu \in \mathbb{N} \Longleftrightarrow uu \in INTEGER \wedge 0 \leq uu \wedge uu \leq MAXINT$$

Substituting:

$$\neg(\{uu \mid uu \in \mathbb{N} \wedge uu \leq ServerVerNo \wedge AFile \mapsto uu \in \text{dom}(ServerRepository)\} = \{\})$$

From the guards of *Commit3* we know:

$AFile \in FILESSET \wedge$
$ServerRepository \neq \{\} \wedge$
$AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge$
$ServerVerNo + 1 \leq MAXINT$

Thus there exists at least one revision of AFile in ServerRepository:

1. $(AFile \in FILESSET \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
   $\wedge ServerRepository \neq \{\} \wedge ServerVerNo + 1 \leq MAXINT)$
   $\Rightarrow \# uu \,.\, (uu \in \mathbb{N} \Rightarrow (AFile \mapsto uu) \in \mathrm{dom}(ServerRepository))$

This satisfies $AFile \mapsto uu \in \mathrm{dom}(ServerRepository)$ in the set comprehension and shows $ServerRepository(AFile \mapsto uu) \neq \{\}$:

2. $(1. \Rightarrow (AFile \mapsto uu) \in \mathrm{dom}(ServerRepository))$
   $\Rightarrow ServerRepository(AFile \mapsto uu) \neq \{\}$

The output of the set comprehension is a natural number:

uu

From 2.:

$ServerRepository(AFile \mapsto uu) \neq \{\}$
$AFile \mapsto uu \in \mathrm{dom}(ServerRepository)$
$True$

Thus:

$\neg(\{uu \mid uu \in \mathbb{N} \wedge uu \leq ServerVerNo \wedge AFile \mapsto uu$
$\in \mathrm{dom}(ServerRepository)\} = \{\})$
$True$

The guards show the requirements of the lemma are always met.

Lemma 2: Atelier B must prove for the *Commit3* operation in file SVN_03 the results returned by the set comprehension are finite in size. That is, $\mathrm{card}(set\ comprehension) \in \mathbb{N}$:

Hypothesis:

$btrue \Rightarrow \{ww \mid ww \in \mathbb{N} \wedge ww \leq ServerVerNo \wedge (FileN \mapsto ww) \in$
$\mathrm{dom}(ServerRepository)\} \cap \mathbb{N} \in FIN(\mathbb{N})$

As in the previous lemma, we look at the guards for *Commit3*:

$Files \subseteq FILESSET \wedge$
$ServerRepository \neq \{\} \wedge$
$Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge$
$ServerVerNo + 1 \leq MAXINT$

This lemma fails because $\mathbb{N}$ items could be under version control and each item may have $\mathbb{N}$ revisions. Using the above definition of $\mathbb{N}$:

$\mathbb{N} \times \mathbb{N} \notin FIN(\mathbb{N})$
$INT \times INT \notin MAXINT$
$False$

Chapter 6

Conclusions

This thesis shows the development of models of a subset of SVN and CVS from specifications found on the web and in one case, experimentation. Multiple refinement steps build upon each other to show complete implementations of Add, Delete, Move, Copy, Commit, Update, Revert and Status. Some operations are have divided responsibilities: Add, Delete, Move and Copy are invoked by the client. CommitAdd, CommitDelete, CommitCopy and CommitMove are called as part of the Commit operation by the server. Later refinements add support for binary files and a pristine local cache for SVN.

Modeling additional features of SVN and CVS varies in difficulty. Commands such as cat, export, import, info, lock, unlock and branching commands are straightforward. Some commands such as blame, resolved and diff require a better definition of a file to have meaning. Similarly, commands like import and export and multi-user functionality are better expressed within a more refined file system. SVN's cleanup command is difficult to implement as one of it's functions is to resume unfinished operations.

Adding support for different kinds of version control systems also varies in difficulty. Enforcement of semantic properties can be added to the Add, Check-In or CommitAdd operations. Intensional versioning can likewise be added to the Check-Out and Update operations. If the intent of evolution is variants or cooperation, changes to the repository structure and operations are both required. Supporting multi-level versioning requires complete replacement of the ServerRepository relation with an N-ary tree and is much more difficult.

Atelier B 4 is approximately one year old as of the writing of this thesis. Two

important notes on it's use are important to keep in mind: By default the predicate prover (PP1) and predicate prover with first level and typing hypothesis (T1) run for only two mintues before giving up. A few three-argument lambda relations containing nested lambda relations required ten to sixteen hours to prove.

Atelier B 4 has a finite rules base. Under some circumstances users may have to extend it's rules to cover new situations. This requires a deep understanding of both logic and Atelier B 4's implementation of it.

Chapter 7

Full Models

7.1 SVN Model

7.1.1 SVN_01

// Base machine, defining sets, opreations and arguments/return values

MACHINE
SVN_01

SETS
FILESSET; *DATA*; *FILECONTENT*

OPERATIONS
Add1(*Files*) =
PRE
Files $\subseteq$ *FILESSET* $\wedge$ *Files* $\neq \{\}$
THEN
skip
END;

CommitAdd1(*Files*) =
PRE
Files $\subseteq$ *FILESSET* $\wedge$ *Files* $\neq \{\}$

THEN
skip
END;

Delete1(*Files*) =
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\}$
THEN
skip
END;

CommitDelete1(*Files*) =
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\}$
THEN
skip
END;

// *Copy takes a relation of the form* $FILESSET \nrightarrow FILESSET$
Copy1(*NewToOldNames*) =
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
THEN
skip
END;

// *CommitCopy takes a relation of the form* $FILESSET \nrightarrow FILESSET$
CommitCopy1(*NewToOldNames*) =
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
THEN
skip
END;

// *Move takes a relation of the form* $FILESSET \nrightarrow FILESSET$

$Move1(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \pfun FILESSET \land NewToOldNames \neq \{\}$
THEN
skip
END;

// *CommitMove takes a relation of the form* $FILESSET \pfun FILESSET$
$CommitMove1(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \pfun FILESSET \land NewToOldNames \neq \{\}$
THEN
skip
END;

$Committed1 \leftarrow Commit1(Files) =$
PRE
$Committed1 \subseteq FILESSET \land Files \subseteq FILESSET \land Files \neq \{\}$
THEN
$Committed1 := Files$
END;

$Updated1 \leftarrow Update1(Files) =$
PRE
$Updated1 \subseteq FILESSET \land Files \subseteq FILESSET \land Files \neq \{\}$
THEN
$Updated1 := Files$
END;

$CheckedOut1 \leftarrow CheckOut1(Files) =$
PRE
$CheckedOut1 \subseteq FILESSET \land Files \subseteq FILESSET$
THEN
$CheckedOut1 := Files$
END;

$Reverted1 \longleftarrow Revert1(FilesToRevertVer) =$
PRE
$Reverted1 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge FilesToRevertVer \neq \{\}$
THEN
$Reverted1 := \mathrm{dom}(FilesToRevertVer)$
END;

// Necessary placeholders for SVN_06

$Status1(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\}$
THEN
skip
END;

$CheckIn1(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\}$
THEN
skip
END
END

7.1.2 SVN_02

MACHINE
SVN_02

INCLUDES
SVN_01

VARIABLES
ClientFiles, *ServerRepository*, *ServerVerNo*

INVARIANT
ClientFiles $\in$ *FILESSET* $\nrightarrow$ *FILECONTENT* // *Can shrink*

$\wedge$ *ServerRepository* $\in$ (*FILESSET* $\times \mathbb{N}$) $\nrightarrow$ *FILECONTENT* // *Only grows*

$\wedge$ *ServerVerNo* $\in \mathbb{N}$ // *Only grows*

// *Invariants*

$\wedge$ ((*ServerRepository* $\neq \{\}$) $\iff$ (*ServerVerNo* > 0))

$\wedge$ ($\forall$ *pp* . (*pp* $\in \mathbb{N} \wedge$ *pp* $\in$ ran(dom(*ServerRepository*))
$\Rightarrow$ *pp* $\leq$ *ServerVerNo*))

// *Don't Prove*
// $\wedge$ (max(ran(dom(*ServerRepository*))) = *ServerVerNo*)

// $\wedge$ ($\forall$ *rr* . (*rr* $\in \mathbb{N}1 \wedge$ *rr* $\leq$ *ServerVerNo*
// $\Rightarrow$ *rr* $\in$ ran(dom(*ServerRepository*))))

INITIALISATION
ClientFiles := {} || *ServerRepository* := {} || *ServerVerNo* := 0

OPERATIONS

$CommitAdd2(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
THEN
$ServerVerNo := ServerVerNo + 1 \parallel$

// *(Invariant) Repository is read only*
$ServerRepository := ServerRepository \cup$
$\lambda\ FileN, Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN \in Files$
$\wedge\ Ver = ServerVerNo + 1 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ FileN \in \mathrm{dom}\ (ClientFiles) \wedge FileN \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
$|$
$ClientFiles(FileN))$
END;

$Delete2(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
THEN
$ClientFiles := Files \lhd ClientFiles$
END;

$Copy2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
THEN
$ClientFiles := ClientFiles \cup$
$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$

$\wedge\ NewToOldNames(NewF) \in \mathrm{dom}(ClientFiles) \wedge NewF \notin \mathrm{dom}(ClientFiles)$
|
$ClientFiles(NewToOldNames(NewF))$
END;

$CommitCopy2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$ // *Files now exist on client*
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{ran}(NewToOldNames) = \{\}$
THEN
$ServerVerNo := ServerVerNo + 1 \parallel$

// *(Invariant) Repository is read only*
$ServerRepository := ServerRepository \cup$
$\lambda\ FileN, Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N}$
$\wedge\ FileN \in \mathrm{dom}(NewToOldNames)$
$\wedge\ (NewToOldNames(FileN), Ver) \in \mathrm{dom}(ServerRepository)$
$\wedge\ (FileN, Ver) \notin \mathrm{dom}(ServerRepository)$
|
$ServerRepository(NewToOldNames(FileN), Ver))$
END;

// *Move takes a relation of the form* $FILESSET \nrightarrow FILESSET$
$Move2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

THEN

$ClientFiles := \mathrm{ran}(NewToOldNames) \lhd ((ClientFiles \cup$
$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$
$\wedge NewToOldNames(NewF) \in \mathrm{dom}(ClientFiles) \wedge NewF \notin \mathrm{dom}(ClientFiles)$
$|$
$ClientFiles(NewToOldNames(NewF)))))$

END;

// CommitMove takes a relation of the form FILESSET ⇸ FILESSET
$CommitMove2(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge \mathrm{ran}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ServerVerNo := ServerVerNo + 1 \parallel$

// (Invariant) Repository is read only
$ServerRepository := ServerRepository \cup$
$\lambda\ FileN, Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N}$
$\wedge FileN \in \mathrm{dom}(NewToOldNames) \wedge Ver \leq ServerVerNo$
$\wedge ServerVerNo + 1 \leq MAXINT$
$\wedge NewToOldNames(FileN) \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge (NewToOldNames(FileN), Ver) \in \mathrm{dom}(ServerRepository)$
$\wedge (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
$|$
$ServerRepository(NewToOldNames(FileN), Ver))$

END;

$Committed2 \longleftarrow Commit2(Files) =$
PRE

$Committed2 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

THEN

$ServerVerNo := ServerVerNo + 1 \parallel$

$ServerRepository := ServerRepository \cup$
$\lambda\ FileN, Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN \in Files$
$\wedge\ Ver = ServerVerNo + 1 \wedge FileN \in \mathrm{dom}(ClientFiles)$
$\wedge\ FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
$|$
$ClientFiles(FileN))$

END; $<$

$Updated2 \longleftarrow Update2(Files) =$

PRE

$Updated2 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

THEN

$ClientFiles := (Files \lhd ClientFiles) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files \wedge FileN \notin \mathrm{dom}(ClientFiles)$
$\wedge\ FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge FileN \notin \mathrm{dom}(ClientFiles)$
$|$
$ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1)) \parallel$
// *Merge if changes are disjoint*
// *User intervention required if changes overlap*

$Updated2 := Files$

END;

$CheckedOut2 \longleftarrow CheckOut2(Files) =$

PRE

$CheckedOut2 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = \{\}$

THEN

$ClientFiles := (Files \lhd ClientFiles) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files$
$\wedge\ FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge FileN \notin \mathrm{dom}(ClientFiles)$
$|$
$ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1)) \parallel$

$CheckedOut2 := Files$

END;

$Reverted2 \longleftarrow Revert2(FilesToRevertVer) =$

PRE

$Reverted2 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ ServerVerNo + 1 \leq MAXINT$

THEN

$ClientFiles := (\mathrm{dom}(FilesToRevertVer) \lhd ClientFiles) \cup$
$\lambda\ FileNm.(FileNm \in FILESSET \wedge FileNm \in \mathrm{dom}(FilesToRevertVer)$
$\wedge\ (FileNm \mapsto FilesToRevertVer(FileNm)) \in \mathrm{dom}(ServerRepository)$
$\wedge\ FileNm \notin \mathrm{dom}(ClientFiles)$
$|$
$ServerRepository(FileNm \mapsto FilesToRevertVer(FileNm))) \parallel$

$Reverted2 := \mathrm{dom}(FilesToRevertVer)$

END

END

7.1.3 SVN_03

MACHINE
SVN_03

INCLUDES
SVN_02

VARIABLES
ClientVersionNo

INVARIANT
$ClientVersionNo \in FILESSET \rightarrow \mathbb{N}$ // *Can shrink*

// *Invariants*

// *Doesn't prove*
// $\wedge$ $(ClientVersionNo \subseteq \mathrm{dom}(ServerRepository))$

INITIALISATION
$ClientVersionNo := \{\}$

OPERATIONS

$CommitAdd3(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
THEN
$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files$
$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$

$|$
$ServerVerNo + 1) \parallel$

$CommitAdd2(Files)$
END;

$Delete3(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge Files \subseteq \mathrm{dom}(ClientFiles)$
THEN
$Delete2(Files)$
END;

$CommitDelete3(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge ClientVersionNo \neq \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
THEN
$ClientVersionNo := Files \lhd ClientVersionNo$
END;

$Copy3(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \rightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
THEN
$Copy2(NewToOldNames)$
END;

$CommitCopy3(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \pfun FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$

$\wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{ran}(NewToOldNames) = \{\}$

THEN

$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$
$\wedge\ NewToOldNames(NewF) \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ NewF \notin \mathrm{dom}(ClientVersionNo)$
$|$
$ClientVersionNo(NewToOldNames(NewF))) \parallel$

$CommitCopy2(NewToOldNames)$

END;

$Move3(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \pfun FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

THEN

$Move2(NewToOldNames)$

END;

$CommitMove3(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \pfun FILESSET \wedge NewToOldNames \neq \{\}$

$\wedge\ ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{ran}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ \forall uu.(uu \in \mathrm{ran}(NewToOldNames) \Rightarrow \#\ vv.(vv \in \mathbb{N} \wedge vv \leq ServerVerNo$
$\wedge\ (uu \mapsto vv) \in \mathrm{dom}(ServerRepository)))$

THEN

$ClientVersionNo := \mathrm{ran}(NewToOldNames) \lhd (ClientVersionNo \cup$
$\lambda\ NewF.(NewF \in FILESSET \wedge NewF \in \mathrm{dom}(NewToOldNames)$
$\wedge\ NewF \notin \mathrm{dom}(ClientVersionNo)$
$\wedge\ NewToOldNames(NewF) \in \mathrm{dom}(ClientVersionNo)$
$|$
$ClientVersionNo(NewToOldNames(NewF)))) \parallel$

$CommitMove2(NewToOldNames)$

END;

$Committed3 \longleftarrow Commit3(Files) =$

PRE

$Committed3 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$

THEN

$ClientVersionNo := (Files \lhd ClientVersionNo) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files \wedge FileN \in \mathrm{dom}(ClientFiles)$
$\wedge\ ClientFiles(FileN) \neq ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq ServerVerNo$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}))$

$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$
|
$ServerVerNo + 1)\ \|$

$Committed3 \longleftarrow Commit2(Files)$
END;

$Updated3 \longleftarrow Update3(Files) =$
PRE
$Updated3 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ClientVersionNo := (Files \lhd ClientVersionNo) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files$
$\wedge\ ServerRepository(FileN \mapsto \max($
$\{xx \mid xx \in \mathbb{N} \wedge xx \leq ServerVerNo$
$\wedge\ (FileN \mapsto xx) \in \mathrm{dom}(ServerRepository)\})) \neq ClientFiles(FileN)$
$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$
|
$\max($
$\{yy \mid yy \in \mathbb{N} \wedge yy \leq ServerVerNo$
$\wedge\ (FileN \mapsto yy) \in \mathrm{dom}(ServerRepository)\}))\ \|$

$Updated3 \longleftarrow Update2(Files)$
END;

$CheckedOut3 \longleftarrow CheckOut3(Files) =$
PRE
$CheckedOut3 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$

THEN

$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN \in Files$
$\wedge FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$\max($
$\{xx \mid xx \in \mathbb{N} \wedge xx \leq ServerVerNo$
$\wedge (FileN \mapsto xx) \in \mathrm{dom}(ServerRepository)\})$
$+ 1) \parallel$

$CheckedOut3 \longleftarrow CheckOut2(Files)$

END;

$Reverted3 \longleftarrow Revert3(FilesToRevertVer) =$

PRE

$Reverted3 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$

THEN

$ClientVersionNo := (\mathrm{dom}(FilesToRevertVer) \lhd ClientVersionNo) \cup$
$\lambda\ FileNm.(FileNm \in FILESSET \wedge FileNm \in \mathrm{dom}(FilesToRevertVer)$
$\wedge (FileNm \mapsto FilesToRevertVer(FileNm)) \in \mathrm{dom}(ServerRepository)$
$\wedge FileNm \notin \mathrm{dom}(ClientVersionNo)$
$|$
$FilesToRevertVer(FileNm)) \parallel$

$Reverted3 \longleftarrow Revert2(FilesToRevertVer)$

END

END

7.1.4 SVN_04

MACHINE
SVN_04

INCLUDES
SVN_03

VARIABLES
ServerUVC

INVARIANT
$ServerUVC \subseteq FILESSET$ // *Can shrink*

// *ServerUVC Invariants*

$\wedge\ (\mathrm{dom}(ClientVersionNo) \subseteq ServerUVC)$

// *Doesn't prove*
// $\wedge\ (ServerUVC \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)))$

INITIALISATION
$ServerUVC := \{\}$

OPERATIONS

$CommitAdd4(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap ServerUVC = \{\}$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
THEN
$ServerUVC := ServerUVC \cup Files \parallel$

$CommitAdd3(Files)$
END;

$Delete4(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
THEN
$Delete3(Files)$
END;

$Copy4(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
THEN
$Copy3(NewToOldNames)$
END;

$CommitCopy4(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerUVC \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap ServerUVC = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{ran}(NewToOldNames) = \{\}$

THEN
$ServerUVC := ServerUVC \cup \mathrm{dom}(NewToOldNames) \parallel$

$CommitCopy3(NewToOldNames)$
END;

$Move4(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
THEN
$Move3(NewToOldNames)$
END;

$CommitMove4(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerUVC \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{ran}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap ServerUVC = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ServerUVC := (ServerUVC \cup \mathrm{dom}(NewToOldNames)) - \mathrm{ran}(NewToOldNames) \parallel$

$CommitMove3(NewToOldNames)$
END;

$Committed4 \longleftarrow Commit4(Files) =$

PRE

$Committed4 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$

THEN

$Committed4 \longleftarrow Commit3(Files)$

END;

$Updated4 \longleftarrow Update4(Files) =$

PRE

$Updated4 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$

THEN

$Updated4 \longleftarrow Update3(Files)$

END;

$CheckedOut4 \longleftarrow CheckOut4(Files) =$

PRE

$CheckedOut4 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ Files \subseteq ServerUVC \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = \{\}$

THEN

$CheckedOut4 \longleftarrow CheckOut3(Files)$

END;

$Reverted4 \longleftarrow Revert4(FilesToRevertVer) =$

PRE

$Reverted4 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$

$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$

THEN

$Reverted4 \longleftarrow Revert3(FilesToRevertVer)$

END

END

7.1.5 SVN_05

MACHINE
SVN_05

INCLUDES
SVN_04

VARIABLES
$ShadowUVC$

INVARIANT
$ShadowUVC \subseteq FILESSET$

INITIALISATION
$ShadowUVC := \{\}$

OPERATIONS
$Add5(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap ServerUVC = \{\} \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ Files \cap ShadowUVC = \{\}$
THEN
$ShadowUVC := ShadowUVC \cup Files$
END;

$Delete5(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge ShadowUVC \neq \{\} \wedge Files \subseteq ShadowUVC$
THEN
$ShadowUVC := ShadowUVC - Files \parallel$

$Delete4(Files)$
END;

$Copy5(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
THEN
$ShadowUVC := ShadowUVC \cup \mathrm{dom}(NewToOldNames) \parallel$

$Copy4(NewToOldNames)$
END;

$Move5(NewToOldNames) =$
PRE
$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\}$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
THEN
$Move4(NewToOldNames) \parallel$

$ShadowUVC := (ShadowUVC \cup \mathrm{dom}(NewToOldNames)) - \mathrm{ran}(NewToOldNames)$
END;

$Committed5 \longleftarrow Commit5(Files) =$
PRE
$Committed5 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$

$\wedge\ ShadowUVC \neq \{\}\ \wedge\ ClientFiles \neq \{\}\ \wedge\ ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\}\ \wedge\ ClientVersionNo \neq \{\}\ \wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)\ \wedge\ Files \subseteq ShadowUVC$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge\ Files \subseteq ServerUVC$
$\wedge\ ServerVerNo > 0\ \wedge\ ServerVerNo + 1 \leq MAXINT$

THEN

$Committed5 \longleftarrow Commit4(Files)$

END;

$Updated5 \longleftarrow Update5(Files) =$

PRE

$Updated5 \subseteq FILESSET\ \wedge\ Files \subseteq FILESSET\ \wedge\ Files \neq \{\}$
$\wedge\ ShadowUVC \neq \{\}\ \wedge\ ClientFiles \neq \{\}\ \wedge\ ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\}\ \wedge\ ClientVersionNo \neq \{\}\ \wedge\ ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT\ \wedge\ Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq ShadowUVC\ \wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge\ Files \subseteq ServerUVC$

THEN

$Updated5 \longleftarrow Update4(Files)$

END;

$CheckedOut5 \longleftarrow CheckOut5(Files) =$

PRE

$CheckedOut5 \subseteq FILESSET\ \wedge\ Files \subseteq FILESSET\ \wedge\ Files \neq \{\}$
$\wedge\ ServerRepository \neq \{\}\ \wedge\ ServerUVC \neq \{\}\ \wedge\ ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT\ \wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ Files \subseteq ServerUVC\ \wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = \{\}\ \wedge\ Files \cap ShadowUVC = \{\}$

THEN

$ShadowUVC := ShadowUVC \cup Files \parallel$

$CheckedOut5 \longleftarrow CheckOut4(Files)$

END;

$Reverted5 \longleftarrow Revert5(FilesToRevertVer) =$

PRE

$Reverted5 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \pfun \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\} \wedge ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT \wedge \mathrm{dom}(FilesToRevertVer) \subseteq ShadowUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$

THEN

$Reverted5 \leftarrow\!\!- Revert4(FilesToRevertVer)$

END

END

7.1.6 SVN_06

MACHINE

SVN_06

INCLUDES

SVN_05

VARIABLES

ShadowRepository, *ShadowVerNo*

INVARIANT

$ShadowRepository \in (FILESSET \times \mathbb{N}) \pfun FILECONTENT$ // *Only grows*

$\land\ ShadowVerNo \in \mathbb{N}$

// ------ *Invariants* ------

$\land\ ((ShadowRepository \neq \{\}) \limp (ShadowVerNo > 0))$

$\land\ (\forall\ pp\ .\ (pp \in \mathbb{N} \land pp \in \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\quad \Rightarrow pp \leq ShadowVerNo))$

// *Don't prove*
// $\land\ (ShadowUVC \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)))$

// $\land\ \mathrm{max}(\mathrm{ran}(\mathrm{dom}(ShadowRepository))) = ShadowVerNo$

// $\land\ \forall rr\ .\ (rr \in \mathbb{N}1 \land rr \leq ShadowVerNo \Rightarrow$
// $\quad rr \in \mathrm{ran}(\mathrm{dom}(ShadowRepository)))$

INITIALISATION

$ShadowRepository := \{\} \parallel ShadowVerNo := 0$

OPERATIONS

$Add6(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap ServerUVC = \{\} \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\ ShadowVerNo \geq 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \cap ShadowUVC = \{\}$
THEN
$ShadowVerNo := ShadowVerNo + 1 \parallel$

$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileNm, Ver.(FileNm \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileNm \in Files$
$\wedge\ Ver = ShadowVerNo + 1 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ (FileNm \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
$\mid$
$ClientFiles(FileNm)) \parallel$

$Add5(Files)$
END;

$Delete6(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq ShadowUVC \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ ShadowVerNo + 1 \leq MAXINT$
THEN
$Delete5(Files)$
END;

$Copy6(NewToOldNames) =$
PRE

$NewToOldNames \in FILESSET \pfun FILESSET \land NewToOldNames \neq \{\}$
$\land\ ClientFiles \neq \{\} \land ShadowUVC \neq \{\} \land ShadowRepository \neq \{\}$
$\land\ ShadowVerNo > 0 \land ShadowVerNo + 1 \leq MAXINT$
$\land\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\land\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\land\ \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\land\ \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
$\land\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\land\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$

THEN

$ShadowVerNo := ShadowVerNo + 1 \parallel$

$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \land Ver \in \mathbb{N}$
$\land\ FileN \in \mathrm{dom}(NewToOldNames) \land Ver < ShadowVerNo + 1$
$\land\ ShadowVerNo + 1 \leq MAXINT$
$\land\ NewToOldNames(FileN) \in \mathrm{dom}(ClientFiles)$
$\land\ (NewToOldNames(FileN),\ Ver) \in \mathrm{dom}(ShadowRepository)$
$\land\ (FileN,\ Ver) \notin \mathrm{dom}(ShadowRepository)$
$|$
$ShadowRepository(NewToOldNames(FileN),\ Ver)) \parallel$

$Copy5(NewToOldNames)$

END;

$Move6(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \pfun FILESSET \land NewToOldNames \neq \{\}$
$\land\ ClientFiles \neq \{\} \land ShadowUVC \neq \{\} \land ShadowRepository \neq \{\}$
$\land\ ShadowVerNo > 0 \land ShadowVerNo + 1 \leq MAXINT$
$\land\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\land\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\land\ \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\land\ \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
$\land\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$

$\wedge\, \mathrm{ran}(\mathit{NewToOldNames}) \subseteq \mathrm{dom}(\mathrm{dom}(\mathit{ShadowRepository}))$

THEN

$\mathit{ShadowVerNo} := \mathit{ShadowVerNo} + 1 \parallel$

$\mathit{ShadowRepository} := \mathit{ShadowRepository} \cup$

$\lambda\ \mathit{FileN},\ \mathit{Ver}.($

$\mathit{FileN} \in \mathit{FILESSET} \wedge \mathit{Ver} \in \mathbb{N} \wedge \mathit{FileN} \in \mathrm{dom}(\mathit{NewToOldNames})$

$\wedge\ \mathit{Ver} < \mathit{ShadowVerNo} + 1 \wedge \mathit{ShadowVerNo} + 1 \leq \mathit{MAXINT}$

$\wedge\ \mathit{NewToOldNames}(\mathit{FileN}) \in \mathrm{dom}(\mathit{ClientFiles})$

$\wedge\ (\mathit{NewToOldNames}(\mathit{FileN}),\ \mathit{Ver}) \in \mathrm{dom}(\mathit{ShadowRepository})$

$\wedge\ (\mathit{FileN},\ \mathit{Ver}) \notin \mathrm{dom}(\mathit{ShadowRepository})$

$|$

$\mathit{ShadowRepository}(\mathit{NewToOldNames}(\mathit{FileN}),\ \mathit{Ver})) \parallel$

$\mathit{Move5}(\mathit{NewToOldNames})$

END;

$\mathit{Committed6} \longleftarrow \mathit{Commit6}(\mathit{Files}) =$

PRE

$\mathit{Committed6} \subseteq \mathit{FILESSET} \wedge \mathit{Files} \subseteq \mathit{FILESSET} \wedge \mathit{Files} \neq \{\}$

$\wedge\ \mathit{ShadowRepository} \neq \{\} \wedge \mathit{ShadowUVC} \neq \{\} \wedge \mathit{ClientFiles} \neq \{\}$

$\wedge\ \mathit{ServerRepository} \neq \{\} \wedge \mathit{ServerUVC} \neq \{\} \wedge \mathit{ClientVersionNo} \neq \{\}$

$\wedge\ \mathit{Files} \subseteq \mathrm{dom}(\mathit{ClientFiles}) \wedge \mathit{Files} \subseteq \mathrm{dom}(\mathit{ClientVersionNo})$

$\wedge\ \mathit{Files} \subseteq \mathrm{dom}(\mathrm{dom}(\mathit{ShadowRepository})) \wedge \mathit{Files} \subseteq \mathit{ShadowUVC}$

$\wedge\ \mathit{Files} \subseteq \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository})) \wedge \mathit{Files} \subseteq \mathit{ServerUVC}$

$\wedge\ \mathit{ShadowVerNo} > 0 \wedge \mathit{ShadowVerNo} + 1 \leq \mathit{MAXINT}$

$\wedge\ \mathit{ServerVerNo} > 0 \wedge \mathit{ServerVerNo} + 1 \leq \mathit{MAXINT}$

$\wedge\ \mathit{Files} \cap \mathrm{dom}(\mathit{ClientFiles}) = \mathit{Files}$

$\wedge\ \mathit{Files} \cap \mathrm{dom}(\mathit{ClientVersionNo}) = \mathit{Files}$

$\wedge\ \mathit{Files} \cap \mathrm{dom}(\mathrm{dom}(\mathit{ShadowRepository})) = \mathit{Files}$

$\wedge\ \mathit{Files} \cap \mathit{ShadowUVC} = \mathit{Files}$

$\wedge\ \mathit{Files} \cap \mathrm{dom}(\mathrm{dom}(\mathit{ServerRepository})) = \mathit{Files}$

$\wedge\ \mathit{Files} \cap \mathit{ServerUVC} = \mathit{Files}$

THEN

$\mathit{ShadowVerNo} := \mathit{ShadowVerNo} + 1 \parallel$

$$
\begin{array}{l}
\quad ShadowRepository := ShadowRepository \cup \\
\quad\quad \lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN \in Files \\
\quad\quad \wedge\ Ver = ShadowVerNo + 1 \wedge ShadowVerNo + 1 \leq MAXINT \\
\quad\quad \wedge\ ClientFiles(FileN) \neq ShadowRepository(FileN \mapsto \max(\\
\quad\quad\quad \{uu \mid uu \in \mathrm{ran}(\mathrm{dom}(ShadowRepository)) \wedge uu \leq ShadowVerNo \\
\quad\quad\quad\quad \wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ShadowRepository)\})) \\
\quad\quad \wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository) \\
\quad\quad | \\
\quad\quad ClientFiles(FileN)) \parallel \\
\\
\quad Committed6 \longleftarrow Commit5(Files) \\
\textbf{END};
\end{array}
$$

$$
\begin{array}{l}
Updated6 \longleftarrow Update6(Files) = \\
\textbf{PRE} \\
\quad Updated6 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\} \\
\quad \wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\} \\
\quad \wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \\
\quad \wedge\ ClientVersionNo \neq \{\} \wedge ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq \\
MAXINT \\
\quad \wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT \\
\quad \wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \\
\quad \wedge\ Files \subseteq ShadowUVC \wedge Files \subseteq \mathrm{dom}(ClientVersionNo) \\
\quad \wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC \\
\quad \wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository)) \\
\textbf{THEN} \\
\quad ShadowVerNo := ShadowVerNo + 1 \parallel \\
\\
\quad ShadowRepository := ShadowRepository \cup \\
\quad\quad \lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge Ver = ShadowVerNo + 1 \\
\quad\quad \wedge\ FileN \in Files \wedge ServerRepository(FileN \mapsto \max(\\
\quad\quad\quad \{uu \mid uu \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge uu \leq ServerVerNo \\
\quad\quad\quad\quad \wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\})) \neq ClientFiles(FileN) \\
\quad\quad \wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)
\end{array}
$$

$$
\begin{array}{l}
\quad | \\
\quad ServerRepository(FileN \mapsto \max(\\
\quad\quad \{uu \mid uu \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge uu \leq ServerVerNo \\
\quad\quad\quad \wedge (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\})))\ \| \\
\\
\quad Updated6 \longleftarrow Update5(Files) \\
\textbf{END};
\end{array}
$$

$$
\begin{array}{l}
CheckedOut6 \longleftarrow CheckOut6(Files) = \\
\textbf{PRE} \\
\quad CheckedOut6 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\} \\
\quad \wedge ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \\
\quad \wedge ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT \\
\quad \wedge ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT \\
\quad \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC \\
\quad \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\} \\
\quad \wedge Files \cap ShadowUVC = \{\} \\
\quad \wedge ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository)) \\
\quad \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \\
\textbf{THEN} \\
\quad ShadowVerNo := ShadowVerNo + 1\ \| \\
\\
\quad ShadowRepository := ShadowRepository \cup \\
\quad\quad \lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge Ver = ShadowVerNo + 1 \\
\quad\quad \wedge ShadowVerNo + 1 \leq MAXINT \wedge FileN \in Files \\
\quad\quad \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \\
\quad\quad \wedge (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository) \\
\quad\quad | \\
\quad\quad ServerRepository(FileN \mapsto \max(\\
\quad\quad\quad \{vv \mid vv \in \mathrm{ran}(\mathrm{dom}(ServerRepository)) \wedge vv \leq ServerVerNo \\
\quad\quad\quad \wedge (FileN \mapsto vv) \in \mathrm{dom}(ServerRepository)\})))\ \| \\
\\
\quad CheckedOut6 \longleftarrow CheckOut5(Files) \\
\textbf{END};
\end{array}
$$

$Reverted6 \longleftarrow Revert6(FilesToRevertVer) =$

PRE

$Reverted6 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \pfun \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ShadowRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ShadowUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$

THEN

$ShadowVerNo := ShadowVerNo + 1 \parallel$

$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileNm,\ Ver.(FileNm \in FILESSET \wedge Ver \in \mathbb{N} \wedge$
$Ver = ShadowVerNo + 1$
$\wedge\ ShadowVerNo + 1 \leq MAXINT \wedge FileNm \in \mathrm{dom}(FilesToRevertVer)$
$\wedge\ (FileNm \mapsto FilesToRevertVer(FileNm)) \in \mathrm{dom}(ServerRepository)$
$\wedge\ (FileNm \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
$|$
$ServerRepository(FileNm \mapsto FilesToRevertVer(FileNm))$
$) \parallel$

$Reverted6 \longleftarrow Revert5(FilesToRevertVer)$

END

END

7.1.7 SVN_07

MACHINE
SVN_07

INCLUDES
SVN_06

SETS
$STATUS = \{Added, Deleted, NoChange, MostRecent, OutOfDate\}$

OPERATIONS

$Add7(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ Files \cap ServerUVC = \{\} \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\ ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ ShadowVerNo \geq 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \cap ShadowUVC = \{\}$
THEN
$Add6(Files)$
END;

$Delete7(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\}$
$\wedge\ Files \subseteq ShadowUVC \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ ShadowVerNo + 1 \leq MAXINT$
THEN
$Delete6(Files)$

END;

Copy7(*NewToOldNames*) =
PRE
NewToOldNames $\in$ *FILESSET* $\nrightarrow$ *FILESSET* $\wedge$ *NewToOldNames* $\neq \{\}$
$\wedge$ *ClientFiles* $\neq \{\}$ $\wedge$ *ShadowUVC* $\neq \{\}$ $\wedge$ *ShadowRepository* $\neq \{\}$
$\wedge$ *ShadowVerNo* > 0 $\wedge$ *ShadowVerNo* $+ 1 \leq$ *MAXINT*
$\wedge$ dom(*NewToOldNames*) $\cap$ dom(*ClientFiles*) $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ dom(*ClientFiles*)
$\wedge$ dom(*NewToOldNames*) $\cap$ *ShadowUVC* $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ *ShadowUVC*
$\wedge$ dom(*NewToOldNames*) $\cap$ dom(dom(*ShadowRepository*)) $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ dom(dom(*ShadowRepository*))
THEN
Copy6(*NewToOldNames*)
END;

Move7(*NewToOldNames*) =
PRE
NewToOldNames $\in$ *FILESSET* $\nrightarrow$ *FILESSET* $\wedge$ *NewToOldNames* $\neq \{\}$
$\wedge$ *ClientFiles* $\neq \{\}$ $\wedge$ *ShadowUVC* $\neq \{\}$ $\wedge$ *ShadowRepository* $\neq \{\}$
$\wedge$ *ShadowVerNo* > 0 $\wedge$ *ShadowVerNo* $+ 1 \leq$ *MAXINT*
$\wedge$ dom(*NewToOldNames*) $\cap$ dom(*ClientFiles*) $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ dom(*ClientFiles*)
$\wedge$ dom(*NewToOldNames*) $\cap$ *ShadowUVC* $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ *ShadowUVC*
$\wedge$ dom(*NewToOldNames*) $\cap$ dom(dom(*ShadowRepository*)) $= \{\}$
$\wedge$ ran(*NewToOldNames*) $\subseteq$ dom(dom(*ShadowRepository*))
THEN
Move6(*NewToOldNames*)
END;

Committed7 $\longleftarrow$ *Commit7*(*Files*) =
PRE
Committed7 $\subseteq$ *FILESSET* $\wedge$ *Files* $\subseteq$ *FILESSET* $\wedge$ *Files* $\neq \{\}$

$\wedge\ ShadowRepository \neq \{\}\ \wedge\ ShadowUVC \neq \{\}\ \wedge\ ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\}\ \wedge\ ServerUVC \neq \{\}\ \wedge\ ClientVersionNo \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)\ \wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))\ \wedge\ Files \subseteq ShadowUVC$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge\ Files \subseteq ServerUVC$
$\wedge\ ShadowVerNo > 0\ \wedge\ ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0\ \wedge\ ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = Files$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = Files$
$\wedge\ Files \cap ShadowUVC = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = Files$
$\wedge\ Files \cap ServerUVC = Files$

THEN

$Committed7 \longleftarrow Commit6(Files)$

END;

$Updated7 \longleftarrow Update7(Files) =$

PRE

$Updated7 \subseteq FILESSET\ \wedge\ Files \subseteq FILESSET\ \wedge\ Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\}\ \wedge\ ShadowUVC \neq \{\}\ \wedge\ ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\}\ \wedge\ ServerUVC \neq \{\}\ \wedge\ ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0\ \wedge\ ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0\ \wedge\ ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles)\ \wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \subseteq ShadowUVC\ \wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge\ Files \subseteq ServerUVC$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$

THEN

$Updated7 \longleftarrow Update6(Files)$

END;

$CheckedOut7 \longleftarrow CheckOut7(Files) =$

PRE

$CheckedOut7 \subseteq FILESSET\ \wedge\ Files \subseteq FILESSET\ \wedge\ Files \neq \{\}$

$\wedge\ ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ Files \cap ShadowUVC = \{\}$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$

THEN

$CheckedOut7 \longleftarrow CheckOut6(Files)$

END;

$Reverted7 \longleftarrow Revert7(FilesToRevertVer) =$

PRE

$Reverted7 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ShadowRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ShadowUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$

THEN

$Reverted7 \longleftarrow Revert6(FilesToRevertVer)$

END;

// $STATUS = \{Added, Deleted, NoChange, MostRecent, OutOfDate\}$

$Stati \longleftarrow Status7(Files) =$

PRE

$Stati \in FILESSET \nrightarrow STATUS \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$

THEN

// Single statement begins here
Stati := (((((

// 1. Added
λ *File1*.(*File1* $\in$ *FILESSET* $\wedge$ *File1* $\in$ *Files* $\wedge$ *File1* $\in$ dom(*ClientFiles*)
$\wedge$ *File1* $\in$ *ShadowUVC* $\wedge$ *File1* $\in$ dom(dom(*ShadowRepository*))
$\wedge$ *File1* $\notin$ dom(*ClientVersionNo*) $\wedge$ *File1* $\notin$ *ServerUVC*
$\wedge$ *File1* $\notin$ dom(dom(*ServerRepository*)) $\wedge$ *File1* $\notin$ dom(*Stati*)
|
Added))

// 3. Deleted
$\cup\lambda$ *File3*.(*File3* $\in$ *FILESSET* $\wedge$ *File3* $\in$ *Files* $\wedge$ *File3* $\notin$ dom(*ClientFiles*)
$\wedge$ *File3* $\notin$ *ShadowUVC* $\wedge$ *File3* $\in$ dom(dom(*ShadowRepository*))
$\wedge$ *File3* $\in$ dom(*ClientVersionNo*) $\wedge$ *File3* $\in$ *ServerUVC*
$\wedge$ *File3* $\in$ dom(dom(*ServerRepository*)) $\wedge$ *File3* $\notin$ dom(*Stati*)
|
Deleted))

// 5. No changes (To download from the server)
$\cup\lambda$ *File5*.(*File5* $\in$ *FILESSET* $\wedge$ *File5* $\in$ *Files* $\wedge$ *File5* $\in$ dom(*ClientFiles*)
$\wedge$ *File5* $\in$ *ShadowUVC* $\wedge$ *File5* $\in$ dom(dom(*ShadowRepository*))
$\wedge$ *File5* $\in$ dom(*ClientVersionNo*) $\wedge$ *File5* $\in$ *ServerUVC*
$\wedge$ *File5* $\in$ dom(dom(*ServerRepository*)) $\wedge$ *File5* $\notin$ dom(*Stati*)
$\wedge$ *ClientVersionNo*(*File5*) = max(
{*v5* | *v5* $\in$ ran(dom(*ServerRepository*)) $\wedge$ *v5* $\leq$ *ServerVerNo*
$\wedge$ (*File5* $\mapsto$ *v5*) $\in$ dom(*ServerRepository*) $\wedge$ *ServerRepository* $\neq$ {}})
$\wedge$ *ClientFiles*(*File5*) = *ServerRepository*(*File5* $\mapsto$ max(
{*v5* | *v5* $\in$ ran(dom(*ServerRepository*)) $\wedge$ *v5* $\leq$ *ServerVerNo*
$\wedge$ (*File5* $\mapsto$ *v5*) $\in$ dom(*ServerRepository*) $\wedge$ *ServerRepository* $\neq$ {}}))
|
NoChange))

$$
\begin{array}{l}
\quad // \ 6.\ \textit{Most recent copy, server is out-of-date WRT local} \\
\quad \cup \lambda\ \textit{File6}.(\textit{File6} \in \textit{FILESSET} \wedge \textit{File6} \in \textit{Files} \wedge \textit{File6} \in \mathrm{dom}(\textit{ClientFiles}) \\
\qquad \wedge \textit{File6} \in \textit{ShadowUVC} \wedge \textit{File6} \in \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository})) \\
\qquad \wedge \textit{File6} \in \mathrm{dom}(\textit{ClientVersionNo}) \wedge \textit{File6} \in \textit{ServerUVC} \\
\qquad \wedge \textit{File6} \in \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) \wedge \textit{File6} \notin \mathrm{dom}(\textit{Stati}) \\
\qquad \wedge \textit{ClientVersionNo}(\textit{File6}) = \max(\\
\qquad\quad \{v6 \mid v6 \in \mathrm{ran}(\mathrm{dom}(\textit{ServerRepository})) \\
\qquad\quad \wedge v6 \leq \textit{ServerVerNo} \wedge (\textit{File6} \mapsto v6) \in \mathrm{dom}(\textit{ServerRepository}) \\
\qquad\quad \wedge \textit{ServerRepository} \neq \{\}\}) \\
\qquad \wedge \textit{ClientFiles}(\textit{File6}) \neq \textit{ServerRepository}(\textit{File6} \mapsto \max(\\
\qquad\quad \{v6 \mid v6 \in \mathrm{ran}(\mathrm{dom}(\textit{ServerRepository})) \\
\qquad\quad \wedge v6 \leq \textit{ServerVerNo} \wedge (\textit{File6} \mapsto v6) \in \mathrm{dom}(\textit{ServerRepository}) \\
\qquad\quad \wedge \textit{ServerRepository} \neq \{\}\})) \\
\qquad | \\
\qquad \textit{MostRecent})) \\
\\
\quad // \ 7.\ \textit{Local copy is out of date (WRT server)} \\
\quad \cup \lambda\ \textit{File7}.(\textit{File7} \in \textit{FILESSET} \wedge \textit{File7} \in \textit{Files} \wedge \textit{File7} \in \mathrm{dom}(\textit{ClientFiles}) \\
\qquad \wedge \textit{File7} \in \textit{ShadowUVC} \wedge \textit{File7} \in \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository})) \\
\qquad \wedge \textit{File7} \in \mathrm{dom}(\textit{ClientVersionNo}) \wedge \textit{File7} \in \textit{ServerUVC} \\
\qquad \wedge \textit{File7} \in \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) \wedge \textit{File7} \notin \mathrm{dom}(\textit{Stati}) \\
\qquad \wedge \textit{ClientVersionNo}(\textit{File7}) < \max(\\
\qquad\quad \{v7 \mid v7 \in \mathrm{ran}(\mathrm{dom}(\textit{ServerRepository})) \\
\qquad\quad \wedge v7 \leq \textit{ServerVerNo} \wedge (\textit{File7} \mapsto v7) \in \mathrm{dom}(\textit{ServerRepository}) \\
\qquad\quad \wedge \textit{ServerRepository} \neq \{\}\}) \\
\qquad | \\
\qquad \textit{OutOfDate})) \ // \ \textit{Single statement ends here} \\
\textbf{END} \\
\textbf{END}
\end{array}
$$

7.1.8 SVN_08

MACHINE
SVN_08

INCLUDES
SVN_07

VARIABLES
ServerBinaryFile, *ServerCOB*

INVARIANT
$ServerBinaryFile \subseteq FILESSET \wedge$ // *Can shrink*

$ServerCOB \subseteq FILESSET$ // *Can shrink, what is checked out binary*

// *Invariants*

// *Doesn't prove*
// $\wedge$ $(ServerCOB \subseteq ServerBinaryFile)$

INITIALISATION
$ServerBinaryFile := \{\} \; || ServerCOB := \{\}$

OPERATIONS

$Add8(Files) =$
PRE
$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge \; Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge \; Files \cap ServerUVC = \{\} \wedge Files \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge \; Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge \; ServerVerNo \geq 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge \; ShadowVerNo \geq 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge \; ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$

$\wedge\ Files \cap ShadowUVC = \{\}$

THEN

// SVN uses binary detection here. We simulate it by making a random
// subset of the input set Files binary.

ANY *IsBinary*

WHERE $IsBinary \subseteq Files$

THEN

$ServerBinaryFile := ServerBinaryFile \cup$
$\{FileN \mid FileN \in FILESSET \wedge FileN \in Files \wedge FileN \in IsBinary$
$\wedge\ FileN \notin ServerBinaryFile\} \parallel$

$Add7(Files)$

END

END;

$Delete8(Files) =$

PRE

$Files \subseteq FILESSET \wedge Files \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\}$
$\wedge\ Files \subseteq ShadowUVC \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ ShadowVerNo + 1 \leq MAXINT$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$

THEN

$ServerBinaryFile := ServerBinaryFile - Files \parallel$
$Delete7(Files)$

END;

$Copy8(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$

$\wedge\, \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\wedge\, \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
$\wedge\, \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\, \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\, \mathrm{dom}(NewToOldNames) \cap ServerBinaryFile = \{\}$
$\wedge\, (\forall\, xx.(xx \in \mathrm{ran}(NewToOldNames) \wedge xx \in ServerBinaryFile$
$\Rightarrow xx \in ServerCOB))$

THEN

$ServerBinaryFile := ServerBinaryFile \cup$
$\{FileNm \mid FileNm \in \mathrm{dom}(NewToOldNames)$
$\wedge\, NewToOldNames(FileNm) \in ServerBinaryFile$
$\wedge\, FileNm \notin ServerBinaryFile\} \parallel$

$Copy7(NewToOldNames)$

END;

$Move8(NewToOldNames) =$

PRE

$NewToOldNames \in FILESSET \nrightarrow FILESSET \wedge NewToOldNames \neq \{\}$
$\wedge\, ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\}$
$\wedge\, ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\, \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\, \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\, \mathrm{dom}(NewToOldNames) \cap ShadowUVC = \{\}$
$\wedge\, \mathrm{ran}(NewToOldNames) \subseteq ShadowUVC$
$\wedge\, \mathrm{dom}(NewToOldNames) \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\, \mathrm{ran}(NewToOldNames) \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\, (\forall\, xx.(xx \in \mathrm{ran}(NewToOldNames) \wedge xx \in ServerBinaryFile$
$\Rightarrow xx \in ServerCOB))$

THEN

// Add new before subtracting old
$ServerBinaryFile := (ServerBinaryFile \cup$
$\{FileNm \mid FileNm \in \mathrm{dom}(NewToOldNames)$
$\wedge\, NewToOldNames(FileNm) \in ServerBinaryFile$
$\wedge\, FileNm \notin ServerBinaryFile\}) - \mathrm{ran}(NewToOldNames) \parallel$

$Move7(NewToOldNames)$
END;

$Committed8 \longleftarrow Commit8(Files) =$
PRE
$Committed8 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge Files \subseteq ShadowUVC$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = Files$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = Files$
$\wedge\ Files \cap ShadowUVC = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = Files$
$\wedge\ Files \cap ServerUVC = Files$

$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$
THEN
$Committed8 \longleftarrow Commit7(Files)$
END;

$Updated8 \longleftarrow Update8(Files) =$
PRE
$Updated8 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(ClientFiles) \wedge Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \subseteq ShadowUVC \wedge Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$

$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$
// *Once binary files are removed from the update set, it isn't empty*
$\wedge\ Files - \{FileB \mid FileB \in Files \wedge FileB \in ServerCOB\} \neq \{\}$

THEN
// *Updates don't occur for binary files*
LET *NotBinary* **BE**
$NotBinary = Files - \{FileB \mid FileB \in Files \wedge FileB \in ServerCOB\}$
IN
IF $NotBinary \neq \{\}$ **THEN**
$Updated8 \longleftarrow Update7(NotBinary)$
END
END

END;

$CheckedOut8 \longleftarrow CheckOut8(Files) =$
PRE
$CheckedOut8 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\} \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ Files \cap ShadowUVC = \{\}$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$
THEN
// *Record whilch files are checked out as binary*
$ServerCOB := ServerCOB \cup$
$\{FileCO \mid FileCO \in Files \wedge FileCO \in ServerBinaryFile$
$\wedge\ FileCO \notin ServerCOB\}\ ||$

$CheckedOut8 \longleftarrow CheckOut7(Files)$

END;

$Reverted8 \longleftarrow Revert8(FilesToRevertVer) =$
PRE
$Reverted8 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ShadowRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ShadowUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ (\forall\ yy.(yy \in \mathrm{dom}(FilesToRevertVer) \wedge yy \in ServerBinaryFile$
$\Rightarrow yy \in ServerCOB))$
THEN
$Reverted8 \longleftarrow Revert7(FilesToRevertVer)$
END
END

7.1.9 SVN_09

MACHINE
SVN_09

INCLUDES
SVN_08

PROMOTES
Add8, Delete8, Move8, Copy8

VARIABLES
ClientPristine

INVARIANT
$ClientPristine \in FILESSET \times \mathbb{N} \nrightarrow FILECONTENT$

// *Invariants*

// *Don't prove*
// $\wedge$ $(\mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)))$

// $\wedge$ $(\mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq ServerUVC)$

// $\wedge$ $(\mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq \mathrm{dom}(ClientVersionNo))$

// $\wedge$ $(\mathrm{dom}(\mathrm{dom}(ClientPristine)) \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)))$

// $\wedge$ $(ServerBinaryFile \subseteq \mathrm{dom}(\mathrm{dom}(ClientPristine)))$

// $\wedge$ $(ServerCOB \subseteq \mathrm{dom}(\mathrm{dom}(ClientPristine))$

INITIALISATION
$ClientPristine := \{\}$

OPERATIONS

$Committed9 \longleftarrow Commit9(Files) =$
PRE
$Committed9 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge Files \subseteq ShadowUVC$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge Files \subseteq ServerUVC$
$\wedge\ ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ \mathrm{card}(ShadowRepository) \in \mathbb{N}$
$\wedge\ Files \cap \mathrm{dom}(ClientFiles) = Files$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = Files$
$\wedge\ Files \cap ShadowUVC = Files$
$\wedge\ Files \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = Files$
$\wedge\ Files \cap ServerUVC = Files$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$
THEN
$Committed9 \longleftarrow Commit8(Files)$
END;

$Updated9 \longleftarrow Update9(Files) =$
PRE
$Updated9 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge ShadowVerNo > 0$
$\wedge\ ShadowVerNo + 1 \leq MAXINT \wedge ServerVerNo > 0$
$\wedge\ ServerVerNo + 1 \leq MAXINT \wedge Files \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge Files \subseteq ShadowUVC$

$\wedge\ Files \subseteq \mathrm{dom}(ClientVersionNo)\ \wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ Files \subseteq ServerUVC\ \wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$
// *Once binary files are removed from the update set, it isn't empty*
$\wedge\ Files - \{FileB \mid FileB \in Files \wedge FileB \in ServerCOB\} \neq \{\}$

THEN

$ClientPristine := ClientPristine \cup$
$\lambda\ FileP,\ VerP.(FileP \in FILESSET \wedge VerP \in \mathbb{N} \wedge FileP \in Files$
$\wedge\ FileP \in \mathrm{dom}(ClientFiles)\ \wedge FileP \in ShadowUVC$
$\wedge\ FileP \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))\ \wedge FileP \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ FileP \in ServerUVC\ \wedge FileP \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ VerP \in \mathrm{ran}(\mathrm{dom}(ServerRepository))$
$\wedge\ FileP \mapsto VerP \in \mathrm{dom}(ServerRepository)$
$\wedge\ FileP \mapsto VerP \notin \mathrm{dom}(ClientPristine)$
$\wedge\ VerP = \max($
$\{vP \mid vP \in \mathbb{N} \wedge vP \leq ServerVerNo$
$\wedge\ (FileP \mapsto vP) \in \mathrm{dom}(ServerRepository)$
$\wedge\ ServerRepository \neq \{\}\})$
$\wedge\ ClientVersionNo(FileP) < VerP$
$|$
$ServerRepository(FileP,\ VerP))\ \|$

$Updated9 \longleftarrow Update8(Files)$

END;

$CheckedOut9 \longleftarrow CheckOut9(Files) =$

PRE

$CheckedOut9 \subseteq FILESSET \wedge Files \subseteq FILESSET \wedge Files \neq \{\}$
$\wedge\ ShadowRepository \neq \{\}\ \wedge ServerRepository \neq \{\}\ \wedge ServerUVC \neq \{\}$
$\wedge\ ShadowVerNo > 0\ \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0\ \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository))\ \wedge Files \subseteq ServerUVC$
$\wedge\ Files \cap \mathrm{dom}(ClientVersionNo) = \{\}\ \wedge Files \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge\ Files \cap ShadowUVC = \{\}$
$\wedge\ ShadowVerNo + 1 \notin \mathrm{ran}(\mathrm{dom}(ShadowRepository))$

$\wedge\ Files \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ (\forall\ xx.(xx \in Files \wedge xx \in ServerBinaryFile \Rightarrow xx \in ServerCOB))$

THEN

$ClientPristine := ClientPristine \cup$
$\lambda\ FileP,\ VerP.(FileP \in FILESSET \wedge VerP \in \mathbb{N} \wedge FileP \in Files$
$\wedge\ FileP \in \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge FileP \in ServerUVC$
$\wedge\ FileP \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ FileP \mapsto VerP \in \mathrm{dom}(ServerRepository)$
$\wedge\ FileP \mapsto VerP \notin \mathrm{dom}(ClientPristine)$
$\wedge\ VerP = \max($
$\{vP \mid vP \in \mathbb{N} \wedge vP \leq ServerVerNo$
$\wedge\ (FileP \mapsto vP) \in \mathrm{dom}(ServerRepository)\})$
$|$
$ServerRepository(FileP,\ VerP))\ \|$

$CheckedOut9 \longleftarrow CheckOut8(Files)$

END;

$Reverted9 \longleftarrow Revert9(FilesToRevertVer) =$

PRE

$Reverted9 \subseteq FILESSET \wedge FilesToRevertVer \in FILESSET \nrightarrow \mathbb{N}1$
$\wedge\ FilesToRevertVer \neq \{\} \wedge \mathrm{dom}(FilesToRevertVer) \subseteq FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ ClientPristine \neq \{\} \wedge ShadowVerNo > 0 \wedge ShadowVerNo + 1 \leq MAXINT$
$\wedge\ ServerVerNo > 0 \wedge ServerVerNo + 1 \leq MAXINT$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ShadowRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ShadowUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientFiles)$
$\wedge\ FilesToRevertVer \subseteq \mathrm{dom}(ServerRepository)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq ServerUVC$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(ClientVersionNo)$
$\wedge\ \mathrm{dom}(FilesToRevertVer) \subseteq \mathrm{dom}(\mathrm{dom}(ClientPristine))$
$\wedge\ \mathrm{ran}(FilesToRevertVer) \cap \mathrm{ran}(\mathrm{dom}(ClientPristine)) = \{\}$
$\wedge\ (\forall\ yy.(yy \in \mathrm{dom}(FilesToRevertVer) \wedge yy \in ServerBinaryFile$

$\Rightarrow yy \in ServerCOB))$
$\wedge (\forall xx.(xx \in \mathrm{ran}(FilesToRevertVer) \Rightarrow xx \leq ServerVerNo))$
THEN
// *Update pristine files*
$ClientPristine := ClientPristine \cup$
$\lambda FileP, VerP.(FileP \in FILESSET \wedge VerP \in \mathbb{N}1 \wedge$
$VerP = ServerVerNo + 1$
$\wedge ServerVerNo + 1 \leq MAXINT \wedge FileP \in \mathrm{dom}(FilesToRevertVer)$
$\wedge FileP \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge (FileP \mapsto FilesToRevertVer(FileP)) \in \mathrm{dom}(ServerRepository)$
$\wedge (FileP \mapsto VerP) \notin \mathrm{dom}(ClientPristine)$
|
$ServerRepository(FileP \mapsto FilesToRevertVer(FileP))$
) ||

$Reverted9 \longleftarrow Revert8(FilesToRevertVer)$
END
END

7.2 CVS Models

7.2.1 CVS_01

CVS_01 is the same as SVN_01. Please refer to the section above.

7.2.2 CVS_02

MACHINE
CVS_02

SETS
FILESSET; *FILECONTENT*

VARIABLES
ClientFiles , *ServerRepository*

INVARIANT
$ClientFiles \in FILESSET \nrightarrow FILECONTENT$ // *Can shrink*

$\wedge\ ServerRepository \in (FILESSET \times \mathbb{N}) \nrightarrow FILECONTENT$ // *Only grows*

INITIALISATION
$ClientFiles := \{\} \parallel ServerRepository := \{\}$

OPERATIONS

$CommitAdd2(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (AFile \mapsto 1) \notin \mathrm{dom}(ServerRepository)$
THEN
$ServerRepository := ServerRepository \cup$
$\{(AFile \mapsto 1) \mapsto ClientFiles(AFile)\}$
END;

$Delete2(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
THEN
$ClientFiles := \{AFile\} \lhd ClientFiles$
END;

$Committed2 \longleftarrow Commit2(AFile) =$
PRE
$Committed2 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge\, ServerRepository \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\, AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ServerRepository := ServerRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN = AFile$
$\wedge\, Ver = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\, (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1$
$\wedge\, FileN \in \mathrm{dom}(ClientFiles) \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\, (FileN \mapsto Ver) \notin \mathrm{dom}(ServerRepository)$
$|$
$ClientFiles(FileN)) \parallel$

$Committed2 := AFile$
END;

$Updated2 \longleftarrow Update2(AFile) =$
PRE
$Updated2 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge\, ServerRepository \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\, AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ClientFiles := (\{AFile\} \lhd ClientFiles) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile \wedge FileN \notin \mathrm{dom}(ClientFiles)$

$\wedge\ FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge FileN \notin \mathrm{dom}(ClientFiles)$
|
$ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}))) \parallel$
// *Merge if changes don' overlap*
// *User intervention required if changes overlap*

$Updated2 := AFile$
END;

$CheckedOut2 \longleftarrow CheckOut2(AFile) =$
PRE
$CheckedOut2 \in FILESSET \wedge AFile \in FILESSET \wedge ServerRepository \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\}$
THEN
$ClientFiles := (\{AFile\} \lhd ClientFiles) \cup$
$\lambda\ FileN.(FileN \in FILESSET$
$\wedge\ FileN = AFile \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ FileN \notin \mathrm{dom}(ClientFiles)$
|
$ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}))) \parallel$

$CheckedOut2 := AFile$
END;

$Reverted2 \longleftarrow Revert2(AFile, AVer) =$
PRE
$Reverted2 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ServerRepository)$
THEN
$ClientFiles := (\{AFile\} \lhd ClientFiles) \cup$

$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile$
$\wedge (FileN \mapsto AVer) \in \mathrm{dom}(ServerRepository)$
$\wedge FileN \notin \mathrm{dom}(ClientFiles)$
$|$
$ServerRepository(FileN \mapsto AVer))\ ||$

$Reverted2 := AFile$

END

END

7.2.3 CVS_03

MACHINE
CVS_03

INCLUDES
CVS_02

VARIABLES
ClientVersionNo

INVARIANT
$ClientVersionNo \in FILESSET \pfun \mathbb{N}$ // *Can shrink*

INITIALISATION
$ClientVersionNo := \{\}$

OPERATIONS

$CommitAdd3(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\, AFile \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\, (AFile \mapsto 1) \notin \mathrm{dom}(ServerRepository)$
$\wedge\, \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
THEN
$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile$
$\wedge\, FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$\max(\{ww \mid ww \in \mathbb{N} \wedge ww + 1 \leq MAXINT$
$\wedge\, (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1) \parallel$

$CommitAdd2(AFile)$
END;

$Delete3(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
THEN
$Delete2(AFile)$
END;

$CommitDelete3(AFile) =$
PRE
$AFile \in FILESSET \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\}$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge AFile \in \mathrm{dom}(ClientVersionNo)$
THEN
$ClientVersionNo := \{AFile\} \lhd ClientVersionNo$
END;

$Committed3 \longleftarrow Commit3(AFile) =$
PRE
$Committed3 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ClientVersionNo := (\{AFile\} \lhd ClientVersionNo) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile \wedge FileN \in \mathrm{dom}(ClientFiles)$
$\wedge ClientFiles(FileN) \neq ServerRepository(FileN \mapsto \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}))$
$\wedge FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$\max(\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1) \parallel$

$Committed3 \longleftarrow Commit2(AFile)$
END;

$Updated3 \longleftarrow Update3(AFile) =$
PRE
$Updated3 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$ClientVersionNo := (\{AFile\} \lhd ClientVersionNo) \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile$
$\wedge\ ServerRepository(FileN \mapsto \max($
$\{xx \mid xx \in \mathbb{N} \wedge xx \leq MAXINT$
$\wedge\ (FileN \mapsto xx) \in \mathrm{dom}(ServerRepository)\}))\ \neq ClientFiles(FileN)$
$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$\max(\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ServerRepository)\}) + 1)\ \|$

$Updated3 \longleftarrow Update2(AFile)$
END;

$CheckedOut3 \longleftarrow CheckOut3(AFile) =$
PRE
$CheckedOut3 \in FILESSET \wedge AFile \in FILESSET \wedge ServerRepository \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\}$
THEN
$ClientVersionNo := ClientVersionNo \cup$
$\lambda\ FileN.(FileN \in FILESSET \wedge FileN = AFile$
$\wedge\ FileN \notin \mathrm{dom}(ClientVersionNo)$
$|$
$\max(\{xx \mid xx \in \mathbb{N} \wedge xx \leq MAXINT$

$\wedge\ (FileN \mapsto xx) \in \mathrm{dom}(ServerRepository)\}) + 1) \parallel$

$CheckedOut3 \longleftarrow CheckOut2(AFile)$

END;

$Reverted3 \longleftarrow Revert3(AFile, AVer) =$

PRE

$Reverted3 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1$

$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ClientVersionNo \neq \{\}$

$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \mapsto AVer \in \mathrm{dom}(ServerRepository)$

$\wedge\ AFile \in \mathrm{dom}(ClientVersionNo)$

THEN

$ClientVersionNo := (\{AFile\} \lhd ClientVersionNo) \cup$

$\lambda\ FileNm.(FileNm \in FILESSET \wedge FileNm = AFile$

$\wedge\ (FileNm \mapsto AVer) \in \mathrm{dom}(ServerRepository)$

$\wedge\ FileNm \notin \mathrm{dom}(ClientVersionNo)$

$|$

$AVer) \parallel$

$Reverted3 \longleftarrow Revert2(AFile, AVer)$

END

END

7.2.4 CVS_04

MACHINE
CVS_04

INCLUDES
CVS_03

VARIABLES
ServerUVC

INVARIANT
$ServerUVC \subseteq FILESSET$ // *Can shrink*

// *ServerUVC Invariants*

$\wedge\ (ServerUVC \subseteq \mathrm{dom}(\mathrm{dom}(ServerRepository)))$

INITIALISATION
$ServerUVC := \{\}$

OPERATIONS

CommitAdd4(*AFile*) =
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \notin \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (AFile \mapsto 1) \notin \mathrm{dom}(ServerRepository) \wedge AFile \notin ServerUVC$
$\wedge\ AFile \notin \mathrm{dom}(ClientVersionNo)$
THEN
$ServerUVC := ServerUVC \cup \{AFile\} \parallel$

CommitAdd3(*AFile*)
END;

$Delete4(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
THEN
$Delete3(AFile)$
END;

$Committed4 \longleftarrow Commit4(AFile) =$
PRE
$Committed4 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
THEN
$Committed4 \longleftarrow Commit3(AFile)$
END;

$Updated4 \longleftarrow Update4(AFile) =$
PRE
$Updated4 \in FILESSET \wedge AFile \in FILESSET \wedge ClientFiles \neq \{\}$
$\wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
THEN
$Updated4 \longleftarrow Update3(AFile)$
END;

$CheckedOut4 \longleftarrow CheckOut4(AFile) =$
PRE
$CheckedOut4 \in FILESSET \wedge AFile \in FILESSET$
$\wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge AFile \in ServerUVC \wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\}$
THEN

$CheckedOut4 \longleftarrow CheckOut3(AFile)$

END;

$Reverted4 \longleftarrow Revert4(AFile, AVer) =$

PRE

$Reverted4 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ServerRepository) \wedge AFile \in ServerUVC$
$\wedge\ AFile \in \mathrm{dom}(ClientVersionNo)$

THEN

$Reverted4 \longleftarrow Revert3(AFile, AVer)$

END

END

7.2.5 CVS_05

MACHINE
CVS_05

INCLUDES
CVS_04

VARIABLES
ShadowUVC

INVARIANT
$ShadowUVC \subseteq FILESSET$

INITIALISATION
$ShadowUVC := \{\}$

OPERATIONS
$Add5(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge \{AFile\} \cap ServerUVC = \{\} \wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge \{AFile\} \cap ShadowUVC = \{\}$
THEN
$ShadowUVC := ShadowUVC \cup \{AFile\}$
END;

$Delete5(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge ShadowUVC \neq \{\} \wedge AFile \in ShadowUVC$
THEN
$ShadowUVC := ShadowUVC - \{AFile\} \parallel$

$Delete4(AFile)$
END;

$Committed5 \longleftarrow Commit5(AFile) =$
PRE
$Committed5 \in FILESSET \wedge AFile \in FILESSET \wedge ShadowUVC \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \in \mathrm{dom}(ClientVersionNo) \wedge AFile \in ShadowUVC$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
THEN
$Committed5 \longleftarrow Commit4(AFile)$
END;

$Updated5 \longleftarrow Update5(AFile) =$
PRE
$Updated5 \in FILESSET \wedge AFile \in FILESSET \wedge ShadowUVC \neq \{\}$
$\wedge\ ClientFiles \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ ClientVersionNo \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
THEN
$Updated5 \longleftarrow Update4(AFile)$
END;

$CheckedOut5 \longleftarrow CheckOut5(AFile) =$
PRE
$CheckedOut5 \in FILESSET \wedge AFile \in FILESSET \wedge ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\} \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ AFile \in ServerUVC \wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\} \wedge \{AFile\} \cap ShadowUVC = \{\}$
THEN
$ShadowUVC := ShadowUVC \cup \{AFile\} \parallel$

$CheckedOut5 \longleftarrow CheckOut4(AFile)$

END;

Reverted5 ⟵ *Revert5*(*AFile*, *AVer*) =
PRE
Reverted5 ∈ *FILESSET* ∧ *AFile* ∈ *FILESSET* ∧ *AVer* ∈ ℕ1
∧ *ShadowUVC* ≠ {} ∧ *ClientFiles* ≠ {} ∧ *ServerRepository* ≠ {}
∧ *ServerUVC* ≠ {} ∧ *ClientVersionNo* ≠ {} ∧ *AFile* ∈ *ShadowUVC*
∧ *AFile* ∈ dom(*ClientFiles*) ∧ *AFile* ↦ *AVer* ∈ dom(*ServerRepository*)
∧ *AFile* ∈ *ServerUVC* ∧ *AFile* ∈ dom(*ClientVersionNo*)
THEN
Reverted5 ⟵ *Revert4*(*AFile*, *AVer*)
END
END

7.2.6 CVS_06

MACHINE
CVS_06

INCLUDES
CVS_05

VARIABLES
ShadowRepository

INVARIANT
$ShadowRepository \in (FILESSET \times \mathbb{N}) \pfun FILECONTENT$ // *Only grows*

// ------ *Invariants* ------

$\land ((ShadowRepository \neq \{\}) \iff (ShadowVerNo > 0))$

$\land (\forall\, pp \cdot (pp \in \mathbb{N} \land pp \in \mathrm{ran}(\mathrm{dom}(ShadowRepository))$
$\Rightarrow pp \leq ShadowVerNo))$

$\land (ShadowUVC \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository)))$

// *Invariants that didn't work*

//$(ServerVerNo + 1 \leq MAXINT)$

// $\land\ (\forall\, cc \cdot (cc \in \mathrm{dom}(ClientVersionNo) \Rightarrow cc \in \mathrm{dom}(\mathrm{dom}(ServerRepository))))$
//-- *Not true b/c of Copy, Move*

// $\land\ (\forall\, aa \cdot (aa \in FILESSET \land aa \in ServerUVC$
/ $\neq >\ aa \in \mathrm{dom}(\mathrm{dom}(ServerRepository)))$))
//-- *Not true b/c of Add*

// $\land\ (ClientVersionNo \neq \{\} \Rightarrow ClientFiles \neq \{\})$

$//-- \textit{Not true b/c of OSDeleteFile}$

$// \ \wedge (\textit{ClientVersionNo} \neq \{\} \iff \textit{ServerUVC} \neq \{\})$
$//-- \textit{Not true b/c of Add}$

$// \ \wedge (\textit{ServerUVC} \neq \{\} \iff \textit{ServerRepository} \neq \{\})$
$//- \textit{Not true b/c of Add}$

$// \textit{Don't prove.}$
$// \wedge \ ((\textit{ShadowVerNo} > 0) \Rightarrow (\max(\mathrm{ran}(\mathrm{dom}(\textit{ShadowRepository}))) = \textit{ShadowVerNo}))$

$// \wedge \ ((\textit{ShadowVerNo} > 0) \Rightarrow (\forall\, rr \,.\, (rr \in \mathbb{N}1 \wedge rr \leq \textit{ShadowVerNo} \Rightarrow$
$//rr \in \mathrm{ran}(\mathrm{dom}(\textit{ShadowRepository}))))$

INITIALISATION

$\textit{ShadowRepository} := \{\}$

OPERATIONS

$\textit{Add6}(\textit{AFile}) =$

PRE

$\textit{AFile} \in \textit{FILESSET} \wedge \textit{ClientFiles} \neq \{\} \wedge \textit{AFile} \in \mathrm{dom}(\textit{ClientFiles})$
$\wedge \{\textit{AFile}\} \cap \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) = \{\}$
$\wedge \{\textit{AFile}\} \cap \textit{ServerUVC} = \{\} \wedge \{\textit{AFile}\} \cap \mathrm{dom}(\textit{ClientVersionNo}) = \{\}$
$\wedge \{\textit{AFile}\} \cap \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository})) = \{\}$
$\wedge \{\textit{AFile}\} \cap \textit{ShadowUVC} = \{\}$

THEN

$\textit{ShadowRepository} := \textit{ShadowRepository} \cup$
$\lambda \ \textit{FileN}, \ \textit{Ver}.(\textit{FileN} \in \textit{FILESSET} \wedge \textit{Ver} \in \mathbb{N} \wedge \textit{FileN} = \textit{AFile}$
$\wedge \ \textit{Ver} = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq \textit{MAXINT}$
$\wedge (\textit{FileN} \mapsto ww) \in \mathrm{dom}(\textit{ShadowRepository})\}) + 1$
$\wedge (\textit{FileN} \mapsto \textit{Ver}) \notin \mathrm{dom}(\textit{ShadowRepository})$
$|$
$\textit{ClientFiles}(\textit{FileN})) \ \|$

$Add5(AFile)$
END;

$Delete6(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge ShadowUVC \neq \{\}$
$\wedge ShadowRepository \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
THEN
$Delete5(AFile)$
END;

$Committed6 \longleftarrow Commit6(AFile) =$
PRE
$Committed6 \in FILESSET \wedge AFile \in FILESSET$
$\wedge ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge AFile \in ShadowUVC$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{AFile\}$
$\wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{AFile\}$
$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{AFile\}$
$\wedge \{AFile\} \cap ShadowUVC = \{AFile\}$
$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{AFile\}$
$\wedge \{AFile\} \cap ServerUVC = \{AFile\}$
THEN
$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge FileN = AFile$
$\wedge Ver = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge (FileN \mapsto ww) \in \mathrm{dom}(ShadowRepository)\}) + 1$
$\wedge ClientFiles(FileN) \neq ShadowRepository(FileN \mapsto \max($
$\{uu \mid uu \in \mathbb{N} \wedge uu \leq MAXINT$
$\wedge (FileN \mapsto uu) \in \mathrm{dom}(ShadowRepository)\}))$

$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
|
$ClientFiles(FileN))$ ||

$Committed6 \longleftarrow Commit5(AFile)$
END;

$Updated6 \longleftarrow Update6(AFile) =$
PRE
$Updated6 \in FILESSET \wedge AFile \in FILESSET \wedge ShadowRepository \neq \{\}$
$\wedge\ ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$

THEN
$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileN, Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge Ver = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ShadowRepository)\}) + 1$
$\wedge\ FileN = AFile \wedge ServerRepository(FileN \mapsto \max($
$\{uu \mid uu \in \mathbb{N} \wedge uu \leq MAXINT$
$\wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\})) \neq ClientFiles(FileN)$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
|
$ServerRepository(FileN \mapsto \max($
$\{vv \mid vv \in \mathbb{N} \wedge vv \leq MAXINT$
$\wedge\ (FileN \mapsto vv) \in \mathrm{dom}(ServerRepository)\}))) \|$

$Updated6 \longleftarrow Update5(AFile)$
END;

$CheckedOut6 \longleftarrow CheckOut6(AFile) =$
PRE

$CheckedOut6 \in FILESSET \wedge AFile \in FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\} \wedge \{AFile\} \cap ShadowUVC = \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$

THEN

$ShadowRepository := ShadowRepository \cup$
$\lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge Ver = \max($
$\{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT$
$\wedge\ (FileN \mapsto ww) \in \mathrm{dom}(ShadowRepository)\}) + 1$
$\wedge\ FileN = AFile \wedge FileN \in \mathrm{dom}(\mathrm{dom}(ServerRepository))$
$\wedge\ (FileN \mapsto \max($
$\{uu \mid uu \in \mathbb{N} \wedge uu \leq MAXINT$
$\wedge\ (FileN \mapsto uu) \in \mathrm{dom}(ServerRepository)\})) \in \mathrm{dom}(ServerRepository)$
$\wedge\ (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository)$
$|$
$ServerRepository(FileN \mapsto \max($
$\{vv \mid vv \in \mathbb{N}$
$\wedge\ vv \leq MAXINT \wedge (FileN \mapsto vv) \in \mathrm{dom}(ServerRepository)))) \parallel$

$CheckedOut6 \longleftarrow CheckOut5(AFile)$

END;

$Reverted6 \longleftarrow Revert6(AFile, AVer) =$

PRE

$Reverted6 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ShadowRepository)$
$\wedge\ AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ServerRepository)$
$\wedge\ AFile \in ServerUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$

THEN

$ShadowRepository := ShadowRepository \cup$

$$
\begin{array}{l}
\quad \lambda\ FileN,\ Ver.(FileN \in FILESSET \wedge Ver \in \mathbb{N} \wedge Ver = \max(\\
\quad\ \{ww \mid ww \in \mathbb{N} \wedge ww \leq MAXINT \\
\qquad \wedge (FileN \mapsto ww) \in \mathrm{dom}(ShadowRepository)\}) + 1 \\
\quad \wedge FileN = AFile \wedge (FileN \mapsto AVer) \in \mathrm{dom}(ServerRepository) \\
\quad \wedge (FileN \mapsto Ver) \notin \mathrm{dom}(ShadowRepository) \\
\quad | \\
\quad ServerRepository(FileN \mapsto AVer) \\
) \parallel \\
\\
\quad Reverted6 \longleftarrow Revert5(AFile, AVer) \\
\textbf{END} \\
\textbf{END}
\end{array}
$$

7.2.7 CVS_07

MACHINE
CVS_07

INCLUDES
CVS_06

SETS
$STATUS = \{Added, Copied, Deleted, Moved, NoChange, MostRecent, OutOfDate\}$

OPERATIONS

$Add7(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge \{AFile\} \cap ServerUVC = \{\} \wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge \{AFile\} \cap ShadowUVC = \{\}$
THEN
$Add6(AFile)$
END;

$Delete7(AFile) =$
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge \{AFile\} \subseteq \mathrm{dom}(ClientFiles)$
$\wedge ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\} \wedge \{AFile\} \subseteq ShadowUVC$
$\wedge \{AFile\} \subseteq \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
THEN
$Delete6(AFile)$
END;

$Committed7 \longleftarrow Commit7(AFile) =$
PRE

$Committed7 \in FILESSET \wedge AFile \in FILESSET \wedge ShadowRepository \neq \{\}$
$\wedge\ ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\} \wedge ServerRepository \neq \{\}$
$\wedge\ ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge AFile \in ShadowUVC$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{AFile\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{AFile\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{AFile\}$
$\wedge\ \{AFile\} \cap ShadowUVC = \{AFile\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{AFile\}$
$\wedge\ \{AFile\} \cap ServerUVC = \{AFile\}$

THEN

$Committed7 \longleftarrow Commit6(AFile)$

END;

$Updated7 \longleftarrow Update7(AFile) =$

PRE

$Updated7 \in FILESSET \wedge AFile \in FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$

THEN

$Updated7 \longleftarrow Update6(AFile)$

END;

$CheckedOut7 \longleftarrow CheckOut7(AFile) =$

PRE

$CheckedOut7 \in FILESSET \wedge AFile \in FILESSET$
$\wedge\ ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\} \wedge \{AFile\} \cap ShadowUVC = \{\}$

$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$

THEN

$CheckedOut7 \longleftarrow CheckOut6(AFile)$

END;

$Reverted7 \longleftarrow Revert7(AFile, AVer) =$

PRE

$Reverted7 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ShadowRepository) \wedge AFile \in ShadowUVC$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \mapsto AVer \in \mathrm{dom}(ServerRepository)$
$\wedge\ AFile \in ServerUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$

THEN

$Reverted7 \longleftarrow Revert6(AFile, AVer)$

END;

// $STATUS = \{Added, Copied, Deleted, Moved, NoChange, MostRecent,$
// $OutOfDate\}$

$Stati \longleftarrow Status7(AFile) =$

PRE

$Stati \in FILESSET \nrightarrow STATUS \wedge AFile \in FILESSET$

THEN

// *Single statement begins here*
$Stati := (((((\ $

// 1. *Added*
$\lambda\ File1.(File1 \in FILESSET \wedge File1 = AFile \wedge File1 \in \mathrm{dom}(ClientFiles)$
$\wedge\ File1 \in ShadowUVC \wedge File1 \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ File1 \notin \mathrm{dom}(ClientVersionNo) \wedge File1 \notin ServerUVC$
$\wedge\ File1 \notin \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge File1 \notin \mathrm{dom}(Stati)$
$|$
$Added))$

$$
\begin{array}{l}
// \ 3.\ \textit{Deleted} \\
\cup\, \lambda\ \textit{File3}.(\,\textit{File3}\ \in \textit{FILESSET}\ \wedge \textit{File3} = \textit{AFile} \\
\quad \wedge\, \textit{File3}\ \notin \mathrm{dom}(\textit{ClientFiles})\ \wedge \textit{File3}\ \notin \textit{ShadowUVC} \\
\quad \wedge\, \textit{File3}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository}))\ \wedge \textit{File3}\ \in \mathrm{dom}(\textit{ClientVersionNo}) \\
\quad \wedge\, \textit{File3}\ \in \textit{ServerUVC}\ \wedge \textit{File3}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) \\
\quad \wedge\, \textit{File3}\ \notin \mathrm{dom}(\textit{Stati}) \\
\quad \mid \\
\quad \textit{Deleted}))
\end{array}
$$

$$
\begin{array}{l}
// \ 5.\ \textit{No changes (To download from the server)} \\
\cup\, \lambda\ \textit{File5}.(\,\textit{File5}\ \in \textit{FILESSET}\ \wedge \textit{File5} = \textit{AFile} \\
\quad \wedge\, \textit{File5}\ \in \mathrm{dom}(\textit{ClientFiles})\ \wedge \textit{File5}\ \in \textit{ShadowUVC} \\
\quad \wedge\, \textit{File5}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository}))\ \wedge \textit{File5}\ \in \mathrm{dom}(\textit{ClientVersionNo}) \\
\quad \wedge\, \textit{File5}\ \in \textit{ServerUVC}\ \wedge \textit{File5}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) \\
\quad \wedge\, \textit{File5}\ \notin \mathrm{dom}(\textit{Stati}) \\
\quad \wedge\, \textit{ClientFiles}(\textit{File5}) = \textit{ServerRepository}(\textit{File5} \mapsto \max(\\
\quad\quad \{v5 \mid v5\ \in \mathbb{N} \wedge v5 \leq \textit{MAXINT}\ \wedge (\textit{File5} \mapsto v5)\ \in \mathrm{dom}(\textit{ServerRepository}) \\
\quad\quad \wedge\, \textit{ServerRepository} \neq \{\}\})) \\
\quad \mid \\
\quad \textit{NoChange}))
\end{array}
$$

$$
\begin{array}{l}
// \ 6.\ \textit{Most recent copy, server is out-of-date WRT local} \\
\cup\, \lambda\ \textit{File6}.(\,\textit{File6}\ \in \textit{FILESSET}\ \wedge \textit{File6} = \textit{AFile} \\
\quad \wedge\, \textit{File6}\ \in \mathrm{dom}(\textit{ClientFiles})\ \wedge \textit{File6}\ \in \textit{ShadowUVC} \\
\quad \wedge\, \textit{File6}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ShadowRepository}))\ \wedge \textit{File6}\ \in \mathrm{dom}(\textit{ClientVersionNo}) \\
\quad \wedge\, \textit{File6}\ \in \textit{ServerUVC}\ \wedge \textit{File6}\ \in \mathrm{dom}(\mathrm{dom}(\textit{ServerRepository})) \\
\quad \wedge\, \textit{File6}\ \notin \mathrm{dom}(\textit{Stati}) \\
\quad \wedge\, \textit{ClientVersionNo}(\textit{File6}) = \max(\\
\quad\quad \{v6 \mid v6\ \in \mathbb{N} \wedge v6 \leq \textit{MAXINT} \\
\quad\quad \wedge\, (\textit{File6} \mapsto v6)\ \in \mathrm{dom}(\textit{ServerRepository}) \\
\quad\quad \wedge\, \textit{ServerRepository} \neq \{\}\}) \\
\quad \wedge\, \textit{ClientFiles}(\textit{File6})\ \neq \textit{ServerRepository}(\textit{File6} \mapsto \max(\\
\quad\quad \{v6 \mid v6\ \in \mathbb{N} \wedge v6 \leq \textit{MAXINT} \\
\quad\quad \wedge\, (\textit{File6} \mapsto v6)\ \in \mathrm{dom}(\textit{ServerRepository})
\end{array}
$$

$\wedge$ *ServerRepository* $\neq \{\}$}))

|

MostRecent))

// 7. *Local copy is out of date (WRT server)*

$\cup \lambda$ *File7*.(*File7* $\in$ *FILESSET* $\wedge$ *File7* = *AFile*

$\wedge$ *File7* $\in$ dom(*ClientFiles*) $\wedge$ *File7* $\in$ *ShadowUVC*

$\wedge$ *File7* $\in$ dom(dom(*ShadowRepository*)) $\wedge$ *File7* $\in$ dom(*ClientVersionNo*)

$\wedge$ *File7* $\in$ *ServerUVC* $\wedge$ *File7* $\in$ dom(dom(*ServerRepository*))

$\wedge$ *File7* $\notin$ dom(*Stati*)

$\wedge$ *ClientVersionNo*(*File7*) $<$ max(

{*v7* | *v7* $\in \mathbb{N} \wedge$ *v7* $\leq$ *MAXINT*

$\wedge$ (*File7* $\mapsto$ *v7*) $\in$ dom(*ServerRepository*)

$\wedge$ *ServerRepository* $\neq \{\}$})

|

OutOfDate)) // *Single statement ends here*

END

END

7.2.8 CVS_08

MACHINE
CVS_08

INCLUDES
CVS_07

VARIABLES
ServerBinaryFile, *ServerCOB*

INVARIANT
$ServerBinaryFile \subseteq FILESSET \wedge$ // *Can shrink*
$ServerCOB \subseteq FILESSET$ // *Can shrink, what is checked out binary*

INITIALISATION
$ServerBinaryFile := \{\} \parallel ServerCOB := \{\}$

OPERATIONS

$Add8(AFile, SpecBinary) =$
PRE
$AFile \in FILESSET \wedge SpecBinary \in BOOL \wedge ClientFiles \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{\}$
$\wedge\ \{AFile\} \cap ServerUVC = \{\} \wedge \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{\}$
$\wedge\ \{AFile\} \cap ShadowUVC = \{\} \wedge AFile \notin ServerBinaryFile$
THEN
ANY *IsBinary*
WHERE $IsBinary \in BOOL$
THEN
// *Incorrect specification*
IF $(IsBinary = TRUE \wedge SpecBinary = FALSE)$
$\vee\ (IsBinary = FALSE \wedge SpecBinary = TRUE)$ **THEN**

skip
ELSE
IF $IsBinary = TRUE \wedge SpecBinary = TRUE$ **THEN**
$ServerBinaryFile := ServerBinaryFile \cup \{AFile\}$
END ||
Add7(*AFile*)
END
END
END;

Delete8(*AFile*) =
PRE
$AFile \in FILESSET \wedge ClientFiles \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\, ShadowUVC \neq \{\} \wedge ShadowRepository \neq \{\} \wedge AFile \in ShadowUVC$
$\wedge\, AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\, ((AFile \in ServerBinaryFile) \Rightarrow (AFile \in ServerCOB))$
THEN
$ServerBinaryFile := ServerBinaryFile - \{AFile\}$ ||
$ServerCOB := ServerCOB - \{AFile\}$ ||
Delete7(*AFile*)
END;

Committed8 ⟵ *Commit8*(*AFile*, *IsBinary*) =
PRE
$Committed8 \in FILESSET \wedge AFile \in FILESSET \wedge IsBinary \in BOOL$
$\wedge\, ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\, ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\, ClientVersionNo \neq \{\} \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\, AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\, AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository)) \wedge AFile \in ShadowUVC$
$\wedge\, AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge\, \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{AFile\}$
$\wedge\, \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{AFile\}$
$\wedge\, \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ShadowRepository)) = \{AFile\}$
$\wedge\, \{AFile\} \cap ShadowUVC = \{AFile\}$

$\wedge \{AFile\} \cap \mathrm{dom}(\mathrm{dom}(ServerRepository)) = \{AFile\}$
$\wedge \{AFile\} \cap ServerUVC = \{AFile\}$
$\wedge ((AFile \in ServerBinaryFile) \Rightarrow (AFile \in ServerCOB))$
THEN
// AFile whose text/binary type is incorrectly specified
IF $(AFile \in ServerBinaryFile \wedge IsBinary \in \{FALSE\})$
$\vee$ $(AFile \notin ServerBinaryFile \wedge IsBinary \in \{TRUE\})$ **THEN**
skip
ELSE
$Committed8 \longleftarrow Commit7(AFile)$
END
END;

$Updated8 \longleftarrow Update8(AFile, IsBinary) =$
PRE
$Updated8 \in FILESSET \wedge AFile \in FILESSET \wedge IsBinary \in BOOL$
$\wedge ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge AFile \in \mathrm{dom}(ClientFiles) \wedge AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge ((AFile \in ServerBinaryFile) \Rightarrow (AFile \in ServerCOB))$
THEN
// AFile whose text/binary type is incorrectly specified
IF $(AFile \in ServerBinaryFile \wedge IsBinary \in \{FALSE\})$
$\vee$ $(AFile \notin ServerBinaryFile \wedge IsBinary \in \{TRUE\})$ **THEN**
skip
ELSE
$Updated8 \longleftarrow Update7(AFile)$
END
END;

$CheckedOut8 \longleftarrow CheckOut8(AFile, IsBinary) =$
PRE
$CheckedOut8 \in FILESSET \wedge AFile \in FILESSET \wedge IsBinary \in BOOL$

$\wedge\ ShadowRepository \neq \{\} \wedge ServerRepository \neq \{\} \wedge ServerUVC \neq \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ServerRepository)) \wedge AFile \in ServerUVC$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientVersionNo) = \{\}$
$\wedge\ \{AFile\} \cap \mathrm{dom}(ClientFiles) = \{\} \wedge \{AFile\} \cap ShadowUVC = \{\}$
$\wedge\ AFile \in \mathrm{dom}(\mathrm{dom}(ShadowRepository))$
$\wedge\ ((AFile \in ServerBinaryFile) \Rightarrow (AFile \in ServerCOB))$

THEN

IF $(AFile \in ServerBinaryFile \wedge IsBinary \in \{FALSE\})$
$\vee\ (AFile \notin ServerBinaryFile \wedge IsBinary \in \{TRUE\})$ **THEN**
skip
ELSE
$CheckedOut8 \longleftarrow CheckOut7(AFile)$
END
END;

$Reverted8 \longleftarrow Revert8(AFile, AVer, IsBinary) =$
PRE
$Reverted8 \in FILESSET \wedge AFile \in FILESSET \wedge AVer \in \mathbb{N}1 \wedge IsBinary \in BOOL$
$\wedge\ ShadowRepository \neq \{\} \wedge ShadowUVC \neq \{\} \wedge ClientFiles \neq \{\}$
$\wedge\ ServerRepository \neq \{\} \wedge ServerUVC \neq \{\} \wedge ClientVersionNo \neq \{\}$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ShadowRepository)$
$\wedge\ AFile \mapsto AVer \in \mathrm{dom}(ServerRepository)$
$\wedge\ AFile \in ShadowUVC \wedge AFile \in \mathrm{dom}(ClientFiles)$
$\wedge\ AFile \in ServerUVC \wedge AFile \in \mathrm{dom}(ClientVersionNo)$
$\wedge\ ((AFile \in ServerBinaryFile) \Rightarrow (AFile \in ServerCOB))$

THEN

IF $(AFile \in ServerBinaryFile \wedge IsBinary \in \{FALSE\})$
$\vee\ (AFile \notin ServerBinaryFile \wedge IsBinary \in \{TRUE\})$ **THEN**
skip
ELSE
$Reverted8 \longleftarrow Revert7(AFile, AVer)$
END
END
END

Bibliography

[Abr96] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meaning.* Cambridge University Press, 1996.

[Abr09] Jean-Raymond Abrial. *Modelling in Event-B: Systems and Software Engineering.* Cambridge University Press, 2009.

[Cle08] ClearSy. *B Language Reference Manual, Ver. 1.8.5*, 2008.

[Cle09] ClearSy. Atelier B 4 web site. *http://www.atelierb.eu/index_en.html*, 2009.

[Col09] Collabnet. Subversion. *http://subversion.tigris.org*, 2009.

[CSFP07] Ben Collins-Sussman, Brian W Fitzpatrick, and C. Michael Pilato. *Version Control With Subversion (For SVN 1.4).* O'Reilly Media, http://svnbook.red-bean.com, 2007.

[CW98] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, 30(2):233—282, 1998.

[FFW09] Leo Frietas, Zheng Fu, and Jim Woocock. POSIX file store in Z/Eves: An experiment in the verified software repository. *Science of Computer Programming*, 74(4):238—257, 2009.

[Fou09] Free Software Foundation. CVS: Concurrent Versions System. *http://www.nongnu.org/cvs*, 2009.

[Gro09] Object Management Group. Uml. *http://www.uml.org/*, 2009.

[Kil97] Tapani Kilpi. New challenges for version control and configuration management: A framework and evaluation. *1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97)*, pages 33—41, 1997.

[Mar06] Dane Marjanovic. Developing a meta-model for release history systems. *University of Zurich, Department of Informatics, Software Evolution and Architecture Lab*, Jan 2006.

[MS84] Carroll Morgan and Bernard Sufrin. Specification of the Unix filing systems. *IEEE Transactions on Software Engineering*, 10(2):128—142, 1984.

[PGC08] Roland Pesch, David G. Grubbs, and Per Cederqvist. *Version Management With CVS (For CVS 1.11.23)*. Free Software Foundation Inc., http://ximbiot.com/cvs/manual, 2008.

[Sch01] Steve Schneider. *The B-Method: An Introduction*. Palgrave, New York, 2001.

[SK99] Emil Sekerinski and Sere Kaisa. *Program Development by Refinement*. Springer-Verlag, London, 1999.

[Spi05a] Diomidis Spinellis. *Version Control, Part 1. IEEE Software*, 22(5):107, 2005.

[Spi05b] Diomidis Spinellis. *Version Control, Part 2. IEEE Software*, 22(6):c3, 2005.

[WG05] D.J. Worth and C. Greenough. *Comparison of CVS and Subversion*. Technical Report RAL-TR-2006-001, Rutherford Appleton Laboratory, Oct 2005.