Technische Universität Wien
Forschungsgruppe Geoinformation
Institut für Geoinformation und Kartographie

Diplomarbeit

Verwendung von Versionsmanagement-Werkzeugen für sicheres kooperatives Arbeiten mit CAD-Plänen

von

Roman Novak

Betreuer: Prof. Dr. Andrew U. Frank

Wien, 2008

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, 14. Juni 2008, Roman Novak

semper et ubique

Inhaltsverzeichnis

Danksagungen

Dank gebührt meiner Frau und meinen Haustieren, die mich während des Schreibens dieser Arbeit betreut, verköstigt und unterhalten haben.

Ich möchte auch Professor Andrew Frank für seine Geduld bis zur Fertigstellung danken. Eine Diplomarbeit neben einer Vollzeitbeschäftigung zu schreiben, ist kräfteraubender als angenommen.

Kurzfassung

Dateibasiertes kooperatives Arbeiten stellt nicht nur in Ingenieurbüros eine Herausforderung dar. Das Einbringen von Änderungen an verschiedenen lokalen Datei-Kopien (speziell Plänen bzw. CAD-Dateien, die Pläne darstellen) in ein zentrales Archiv führt zu Situationen, in denen entschieden werden muss, welche Änderungen den letzten, gültigen Stand darstellen sollen.

Es wird nun angenommen, dass Versionsmanagement Werkzeuge (VCS , Version Control System) zu einer Lösung des genannten Problems beitragen können und es wird untersucht, ob es Ansätze gibt, um solche Entscheidungssituationen effizient handhaben und auflösen zu können.

Anhand eines einfachen Open-Source Tools (OooSVN) wird gezeigt, welche Möglichkeiten es zur Konfliktresolution gibt und welche Prinzipien angewendet werden. Eine Gegenüberstellung von Versionsmanagement-Werkzeugen wie SVN, CVS und DARCS, GIT, die auf verschiedenen Philosophien basieren, leitet zu einer intensiveren Analyse der Versionsverwaltung DARCS über. Die Patch-Theorie, die hinter DARCS steht, wird beleuchtet und es wird hinterfragt, ob diese zur Problemlösung beitragen kann.

Es wird die Anwendbarkeit von Konfliktresolution betrachtet und wie Benutzer bei der Auflösung derselben unterstützt werden können. Die Ausführungen münden in der Entwicklung eines Plugins, welches das kooperative Bearbeiten von CAD-Plänen unterstützt.

Abstract

File based cooperative effort does not only pose a challenge in engineer's offices. Applying changes to various local files and commiting these changes to a central repository leads to conflicting situations, where decisions have to be made, which changes should represent the final valid state.

It is assumed, that version control systems (VCS) can help solving the described problem above. It will be examined, if there is a viable way to support decisions efficiently and solve those conflicting situations.

On the basis of a simple open source tool (OooSVN) it will be demonstrated, which possibilities for conflict resolution exist and which principles are applied. A comparison of version control systems like SVN, CVS and DARCS, GIT, which are based on different underlying philosophies, leads up to a more intensive analysis of the version control system DARCS. The patch theory behind DARCS will be explained and it will be investigated, if it can solve the problems above.

The applicability of conflict resolution in management of technical data respresenting technical drawings will be regarded and how users can be supported in that aspect. The remarks lead to the development of a plugin, which supports the cooperative editing of technical drawings.

Kapitel 1

Einleitung

1.1 Problemstellung: Dateibasierte Kooperation

Im Rahmen von gemeinsamen Anstrengungen unter Personen entsteht der Bedarf, gleichzeitig Bearbeitungen an einer Datei durchzuführen. Problematisch wird die Situation, wenn die korrekte Reihenfolge durch unterschiedliche Personen eingebrachter Änderungen nicht ohne weiteres entscheidbar ist.

Erschwert wird die Situation durch die Verwendung von binären Quasi-Standardformaten, deren Spezifikation nicht offengelegt ist. So gibt es beispielsweise in den Bereichen CAD und Textverarbeitung Anwendungen (AutoCAD[47] und Microsoft Word), die allgemein aufgrund ihrer besonderen Eignung zur Lösung der Aufgabenstellung und ihrer Verbreitung wegen verwendet werden. Die Dateiformate, die dabei benutzt werden, sind für den jeweiligen Bereich optimiert und historisch gewachsen. Historisch gewachsene Formate liegen oftmals nicht als gut definierte, anerkannte Standards vor. Dies stellt dann in der automatisierten Verarbeitung dieser Daten zum Zwecke eines Abgleiches eine Herausforderung dar.

Im Rahmen der Softwareentwicklung haben sich Versionsmanagement-Werkzeuge (VCS) bereits etablieren können. Im Falle von gemeinsamer Bearbeitung von Text-Dateien (z.B. Java Quellcode) sind teilweise effiziente Möglichkeiten zur Konfliktresolution verfügbar. Wie in weiterer Folge illustriert werden wird, gibt es jedoch Konfliktsituationen, die auch bereits mit Text-Dateien schwierig handhabbar sind. Vor diesem Hintergrund wird das VCS DARCS[4] genauer untersucht, konkret ob spezifische Problematiken durch DARCS besser unterstützt werden, als durch andere VCS (s. Kapitel 4). Betrachtet werden auch Ansätze, um kooperatives Arbeiten mit komplexeren Dateiformaten zu unterstützen.

Beispiele für oben erwähnte, weitverbreitete Dateiformate wären die AutoCAD Dateiformate DWG bzw. DXF und das Microsoft Winword Dateiformat

(in zwei Varianten: als binäres Format und als Office Open XML[49]).

Bei den genannten Formaten (man könnte die Liste der Formate sicher fortsetzen) wird eine manuelle, durch einen Anwender durchgeführte Vereinigung relativ rasch aufwändig. Deswegen wird hier der Idee nachgegangen, die dateibasierte Kooperation - speziell im Zusammenhang mit CAD-Plänen - durch VCS zu unterstützen. VCS bieten unter anderem Mechanismen wie Dateisperren, Differenzenbildung und Zusammenführung für Dateien an.

1.2 CAD-Pläne: Situation in Ingenieurbüros

Versionierung und kooperatives Arbeiten im Zusammenhang mit CAD-Plänen sind in Ingenieurbüros ein wichtiges Thema: CAD-Pläne können unter Umständen einen grossen Umfang annehmen (sowohl bezüglich der Dateigrösse, als auch bezüglich des vom CAD-Plan abgebildeten Bereiches). Wenn mehrere Bearbeiter Änderungen an einer Datei durchführen, entstehen dadurch Konfliktsituationen beim Übertragen der neuen Versionen in ein Archiv, die sich schwer auflösen lassen. Dies kann eine zeitaufwändige und mühselige Tätigkeit sein.

Eine Konfliktsituation kann beispielsweise wie folgt entstehen: Zwei Bearbeiter beziehen die gleiche Version eines CAD-Planes aus dem (elektronischen) Archiv. Beide führen Änderungen an dem CAD-Plan durch. Ein Bearbeiter löscht ein Objekt und legt den CAD-Plan im Archiv ab. Ein Bearbeiter führt Änderungen an genau dem Objekt durch, das durch den anderen gelöscht worden ist. Beim Übertragen des CAD-Planes in das Archiv durch den zweiten Bearbeiter entsteht nun folgendes Problem: welche Version soll nun den letzten, gültigen Stand darstellen? Im Falle von CAD-Plänen ist eine Visualisierung der Unterschiede notwendig, um eine Entscheidung treffen zu können.

Die Ausführungen in den folgenden Kapiteln leiten von der allgemeinen Problemstellung über zu einem konkreten Ansatz für die Unterstützung von verteilter Bearbeitung von CAD-Plänen. Es wird ein Plugin vorgestellt, welches den folgenden Arbeitsablauf (Abb. 1.1) unterstützt. Die Abbildung illustriert den Ablauf bei der Vereinigung von Änderungen an zwei verschiedenen Versionen einer Datei. Das Plugin visualisiert die aufgefundenen Unterschiede der im VCS abgelegten Datei-Versionen. Damit wird der Bearbeiter bei Bewertung der Änderungen unterstützt. Er kann sehen, wann und durch wen (via Funktionalitäten des VCS), welche Änderungen (mittels des Plugins) durchgeführt worden sind.

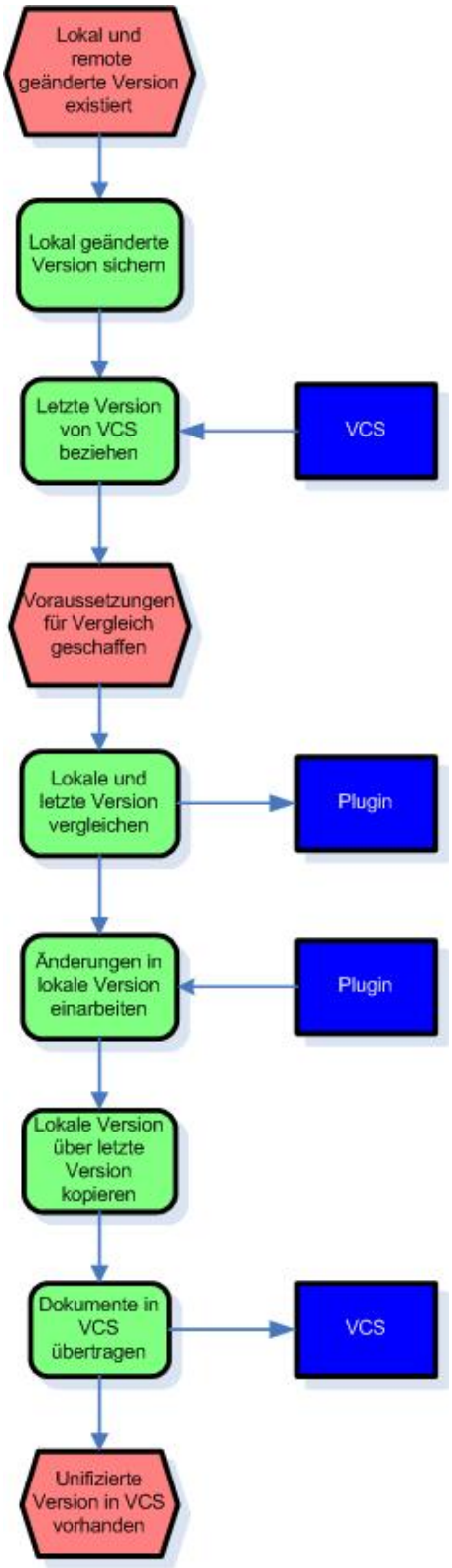

Abbildung 1.1: Workflow: Auflösung Konfliktsituationen mit Hilfe eines Plugins und eines VCS

Kapitel 2

Bestehende Ansätze für dateibasiertes kooperatives Arbeiten

2.1 Organisatorische Ansätze

Es gibt einige organisatorische Ansätze, die kooperatives Arbeiten erleichtern. Diese werden hier kurz diskutiert, da eine rein technische Betrachtung des Problemes der realistisch in der Arbeitswelt anzutreffenden Situation nicht gerecht wird.

2.1.1 Partitionierung einer Aktivität

Partitionierung (lat. *Partitio*, Einteilung) einer Aktivität bedeutet hier eine Zerlegung in einzelne Tätigkeiten, die keinen gleichzeitigen Zugriff auf Resourcen benötigt. Wenn eine solche Einteilung möglich ist, stellt dies eine sehr einfache Art und Weise dar, dateibasiertes kooperatives Arbeiten zu ermöglichen.

Es kommt oft allerdings zu Störungen (z.B: krankheitsbedingte Ausfälle, aktualisierte Priorisierungen von beteiligten Personen), die eine solche Einteilung erschweren.

2.1.2 Verantwortlichkeiten

Eine rein organisatorische Lösung ist die Nominierung eines Verantwortlichen für ein Ergebnis. Wie dieses erreicht wird, ist dem Verantwortlichen überlassen. Diese Vorgehensweise vereinfacht die Sicht auf die Problematik kooperativen Arbeitens. In Wirklichkeit werden sich dahinter weitere organisatorische bzw. technische Ansätze verbergen.

2.1.3 Benachrichtigungen

Benachrichtigungen verbessern die Zusammenarbeit. Dadurch kann sich die Übergabe von einem Bearbeiter zum nächsten effizienter gestalten bzw. es kann auf diverse Zustände der Bearbeitung hingewiesen werden.

Benachrichtigungen werden üblicherweise in Arbeitsabläufen zur Koordination eingesetzt und sind technisch gut unterstützt.

2.1.4 Definierte Abläufe

Definierte Abläufe bei der Bearbeitung von Aufgaben sind hilfreich, um gemeinsame Anstrengungen zu koordinieren: Durch eine Einteilung eines Ablaufes in Aktivitäten, die durch Organisationseinheiten (s. auch z.B. [20]) erledigt werden, lässt sich der Zugriff auf eine Resource (in diesem Falle Dateien) regeln. Im Grunde gibt ein Ablauf den oben genannten Massnahmen einen Rahmen.

Es kann in komplexen Umgebungen allerdings vorkommen, dass Zugriffe auf eine Resource nicht exklusiv gewährleistet werden können. Solange nur ein einziger schreibender Zugriff involviert ist, lässt sich mittels Dateiversionierung bzw. Check-Out Mechanismen (s. später in diesem Kapitel) auch dies erreichen.

Eine strikte Einteilung von Abläufen kann daher die Handhabbarkeit (Transparenz) und Effizienz (Durchlaufzeit) im Umgang mit kritischen Resourcen erhöhen.

Die obige Auflistung dient der Illustration der organisatorischen Möglichkeiten. Die genannten Maßnahmen können rein organisatorisch durchgesetzt werden bzw. technisch unterstützt werden. Auf die technischen Möglichkeiten wird im nächsten Abschnitt eingegangen.

Für Benutzer besonders vorteilhaft wäre eine sofortige Benachrichtigung, wenn sich eine Datei, die man selbst gerade bearbeitet, durch einen anderen Benutzer geändert hat. Dadurch könnte der Benutzer seine eigenen Änderungen so schnell wie möglich mit der geänderten Datei vergleichen. Je später diese Information verfügbar ist, desto mehr Änderungen sind zu vergleichen.

2.2 Technische Ansätze

Die folgenden Beispiele technischer Ansätze für dateibasiertes kooperatives Arbeiten sorgen für eine bessere Umsetzbarkeit der organisatorischen Maßnahmen. Besonderen Ansätzen wie *Visualisierung von Differenzen* und *spezielle Patching Methodiken* wird in späteren Kapiteln mehr Platz eingeräumt.

2.2.1 Workflows

Auf technischer Seite können Abläufe durch Workflowsysteme unterstützt werden. Für dateibasiertes kooperatives Arbeiten kann dies beispielsweise folgendes bedeuten:

- Ein Review einer Datei, bevor diese akzeptiert wird.
- Ein automatisiertes Verschieben einer Datei in ein anderes (serverseitiges) Verzeichnis, auf welches man eventuell sogar keinen Zugriff mehr hat. Dies kann zum Beispiel eine Freigabe der Datei bedeuten bzw. einen Schritt vorwärts in einem Ablauf.
- Verschiedenste Aktionen, die mit einer Datei durchgeführt werden können: Weiterleitung, Transformation, Export, uvm.

2.2.2 Locking

Ein wichtiger Mechanismus für verteiltes Arbeiten ist Locking. Locking bedeutet das Sperren von Resourcen in einem Repository, sodass kein anderer Benutzer Änderungen schreiben kann.

2.2.3 Versionierung

Die Definition des Begriffes Version aus dem Bereich Softwareentwicklung lautet wie folgt [55]:

> In der Softwareentwicklung ist die Version ein definierter Stand einer Software mit allen dazugehörigen Komponenten. Verschiedene Versionen stellen die Veränderung und Weiterentwicklung einer Software oder eines Teils (z.B. Programmbibliothek) über die Zeit dar. Somit werden Systeme zur Versionierung genutzt, um neuere Ausgaben einer Software von einer älteren zu unterscheiden.
>
> Verschiedene numerische, alphanumerische oder auch datumsbasierte Versionsnummern werden zur Unterscheidung und Bezeichnung herangezogen. Der Prozess wird oft durch ein Versionsverwaltungssystem technisch begleitet.

Versionierung ermöglicht die Wiederherstellung vorheriger Dateizustände. Es gibt verschiedene Strategien, um Versionierung vor allem von binären Dateien durchzuführen. Darauf wird hier aber nicht näher eingegangen.

2.2.4 Merging

Unter Merging versteht man das Zusammenführen von verschiedenen Dateiversionen. Dazu kann es kommen, wenn beispielsweise zwei Personen von einem

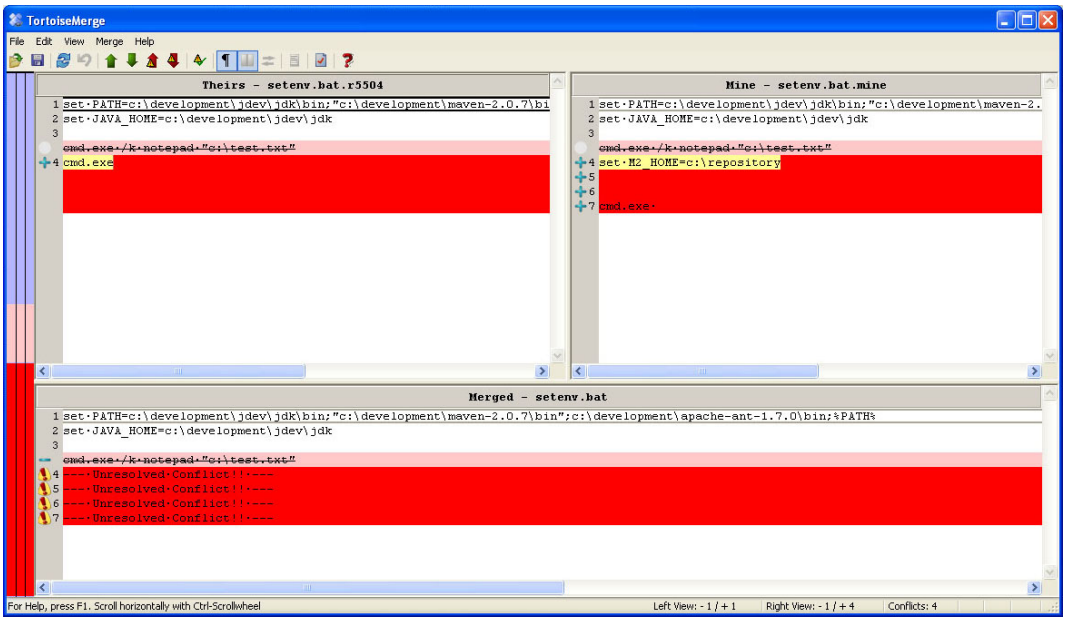

Abbildung 2.1: Visualisierung von Differenzen - Konflikte editieren mit SVN

identen Zustand ausgehend, jeweils lokale Änderungen anbringen und diese dann in das Repository einbringen wollen.

Beispiel SVN: Bei einem Übertragen in das VCS wird der Benutzer darauf hingewiesen, dass sich die Daten zwischenzeitlich geändert haben und eine Aktualisierung des lokalen Bereiches durchgeführt werden sollte. Wenn dieses Update durchgeführt wird, wird versucht, die Änderungen seit dem letzten abgeglichenen Stand in der lokalen Version zu vereinigen (="Merge").

Diese Merge-Operation führt manchmal nicht zum gewünschten Resultat und das Ergebnis sollte jedenfalls geprüft werden, da die semantische Korrektheit zerstört werden kann. Eine Automatisierbarkeit an dieser Stelle wäre bis zu einem gewissen Grad erreichbar, wenn die Semantik des Ergebnisses der Vereinigung automatisierbar überprüfbar ist.

2.2.5 Visualisierung von Differenzen

Die Visualisierung von Differenzen hängt eng mit den Themen Versionierung und vor allem mit Merging zusammen: nicht einfach entscheidbare Konflikte, die beim Merging entstehen, werden erst visualisiert und dem menschlichen Bearbeiter zur Auflösung überlassen. Wie man in Abb. 2.1 sehen kann, entstehen schon beim Editieren weniger Zeilen Konstellationen, die nicht ohne weiteres automatisiert vereinigt werden können (in diesem Falle wurde von zwei Seiten jeweils an der selben Stelle eine bzw. mehr Zeilen eingefügt).

Als technisches Hilfsmittel ist die Visualisierung von Differenzen unerlässlich,

um durch VCS (oder auch DMS) unterstütztes dateibasiertes kooperatives Arbeiten zu ermöglichen. Hierbei ist die Visualisierung der Differenzen in der für die betroffene Datei **nativen** Anwendungsumgebung sehr stark zu empfehlen.

Kapitel 3

VCS und DMS

3.1 Version Control Systems

In den Büchern zum Thema Versionskontrolle werden verschiedenste, sinngemäß gleichlautende Definitionen gegeben.

Die kürzeste Definition des Begriffes Version Control dürfte sein:

> Version control is the art of managing changes to information[9, Kapitel 1].

Eine andere Definition lautet:

> A version control system is a place to store all the various revisions of the stuff you write while developing an application ([16, Seite 8] und [30, Seite 9]).

Die folgende Definition streicht den Teamaspekt, der in dieser Arbeit ein wichtiges Thema darstellt, heraus:

> Version control provides a way to manage the complexity of working with a team of people on a specific project [11, Seite 1].

Eine umfangreichere Beschreibung, die alle vorigen Punkte mitberücksichtigt, gibt Wikipedia [56]:

> Unter einer Versionsverwaltung versteht man ein System, welches typischerweise in der Softwareentwicklung zur Versionierung und um den gemeinsamen Zugriff auf Quelltexte zu kontrollieren, eingesetzt wird. Hierzu werden alle laufenden Änderungen erfasst und alle Versionsstände der Dateien in einem Archiv mit Zeitstempel und Benutzerkennung gesichert. Die Versionsverwaltung ist eine Form des

> Variantenmanagements. Die übergreifende Disziplin ist das Software Configuration Management (kurz: SCM).
>
> Es wird sichergestellt, dass jeder Benutzer mit dem aktuellen Stand arbeitet oder auf Wunsch auf die archivierten Stände zugreifen kann. Dadurch ist eine Versionsverwaltung nicht nur für professionelle Entwickler in großen Teams, sondern auch für einzelne Entwickler interessant. Es kann jederzeit eine ältere Version aufgerufen werden, falls eine Änderung nicht funktioniert und man sich nicht mehr sicher ist, was nun alles geändert wurde.
>
> ...
>
> Für Versionsverwaltungssysteme sind die Abkürzungen VCS (Version Control System) oder SCM (Source Code/Control Managementsystem) gebräuchlich.

Wie man sieht, ist der Begriff VCS ist relativ gut definiert.

3.2 Dokumentenmanagentsysteme

Die folgende Definition von Dokumentenmanagement ist Wikipedia entnommen [48]:

> Dokumentenmanagement dient der datenbankgestützten Verwaltung elektronischer Dokumente. Man versteht darunter in Deutschland die Verwaltung ursprünglich meist papiergebundener Dokumente in elektronischen Systemen. Bei der Verwaltung von Papierdokumenten spricht man dagegen von Schriftgutverwaltung. Zur besseren Unterscheidung wird häufig auch der Begriff EDM Elektronisches Dokumentenmanagement (Electronic Document Management) verwendet. Die Abkürzung DMS steht für Dokumenten-Management-System und wird in einem erweiterten Sinn als Branchenbezeichnung verwendet. Im Amerikanischen steht "Document Management"dagegen begrifflich eingeschränkter für die Verwaltung von Dateien mit Checkin/Checkout, Versionierung und anderen Funktionen. Inzwischen gilt Dokumentenmanagement als eine Komponente des übergreifenden ECM Enterprise Content Management.

Es folgt nun die Definition von DMS von Nix et al. [17, Seite 111]:

> Unter Document Management ist hier nicht die Branchenbezeichnung wie in Deutschland zum Beispiel DMS zu verstehen, sondern eher die Dokumentenmanagementsysteme im "klassischen"oder "engeren"Sinn. Die Aufgabe dieser Systeme ist es, den Lebenszyklus der

Dokumente von der Entstehung bis zur Langzeitarchivierung zu kontrollieren. Zum Document Management gehören unter anderem folgende Funktionen:

- Checkin / Checkout: zur Kontrolle der Konsistenz der gespeicherten Informationen
- Versionsmanagement: zur Kontrolle unterschiedlicher Stände gleicher Information mit Versionen, Revisionen und Renditionen (gleiche Informationen in einem unterschiedlichen Format)
- Suchen und Navigieren: zum Auffinden von Informationen in Strukturen wie virtuellen Akten, Verzeichnissen und Übersichten

Die Funktionen des Document Management überschneiden sich jdeoch zunehmend mit denen der anderen "Manage"-Komponenten, der immer weiter ausgreifenden Funktionalität von Office-Anwendungen wie Outlook/Exchange oder Notes/Domino und den Eigenschaften von "Library Services"zur speichertechnischen Verwaltung der Informationen.

3.3 Unterschiede und Gemeinsamkeiten

VCS und DMS haben in gewissen Bereichen sehr ähnliche Funktionalitäten. Für die Unterstützung kooperativen Arbeitens mit CAD-Plänen werden folgende Anforderungen gestellt:

- Dokumente sollten sicher abgelegt sein. Dies wird durch ein Repository gewährleistet, welches regelmässig gesichert werden sollte.
- Dokumente sollten versioniert sein, damit ein vorheriger Stand bei Bedarf abgerufen werden kann.
- Dokumente sollten für Bearbeitung exklusiv gesperrt werden können.
- Dokumentversionen sollten einfach vereinigt werden können. Diese Forderung ist sehr wichtig, da hier am meisten Aufwand für den Benutzer entsteht.

Tabelle 3.1 listet iausgewählte Funktionalitäten von VCS und DMS. Hierbei sind besonders obige Anforderungen zu beachten.

Repository VCS sind üblicherweise dateibasiert. Subversion (ein VCS) speichert die Daten optional in einer Berkeley DB[36], wobei die empfohlene

Feature	VCS / DMS
Repository	beide
Versionierung	beide
Locking (Checkin/Checkout)	beide
Security	beide
Workflow	DMS
Merging	VCS
Patching	VCS
Visualisierung	beide
Suchen/Navigieren	beide
Collaboration	DMS

Tabelle 3.1: Kurzvergleich VCS und DMS

Konfiguration dateibasiert ist. Die dateibasierte Konfiguration ist die Standardeinstellung seit Version 1.2 (s. auch [10, Kapitel 5, Choosing a Data Store]). Im Falle von DMS wird meistens eine Datenbank benutzt. Hier kann allerdings die Datenhaltung der Metadaten und der eigentlichen Dateien unterschiedlich sein. Aus Performancegründen liegen letzere oft direkt im Dateisystem. DMS verwalten weiters einen (Volltext) Suchindex, der ausserhalb einer Datenbank liegt.

Versionierung (Checkin/Checkout) Versionierung ist die Hauptaufgabe eines VCS. Im Falle von DMS kann Versionierung optional aktiviert werden (z.B. Alfresco, s. auch 3.5.1 für mehr Details).

Ein Check-Out im Kontext von VCS bedeutet das Übertragen von Dateien von einem Repository in einen lokalen Bereich (und hat für sich genommen keinen positiven Einfluss auf gleichzeitige Bearbeitungen). Es können in dem lokalen Bereich Änderungen vorgenommen werden, die dann mit einem Check-In (bzw. Commit) wieder zurück übertragen werden. Es wird dadurch eine neue Version erstellt.

Ein Check-Out im Kontext von DMS bereitet eine neue Version vor. Es wird eine Arbeitskopie erstellt (s. Abbildung 3.1). Die Originaldatei wird damit für Schreiben gesperrt (Locking). Es kann üblicherweise nur ein Check-Out pro Resource durchgeführt werden. Mit einem Check-In wird das Ergebnis wieder in das System übertragen und die Sperre aufgehoben (s. auch [26, Seite 426]: “checkin: Removes the lock on a checked out object and saves it in the Docbase as a new version.”).

Locking Locking sorgt für kontrollierte Schreib-Zugriffe auf Dateien und vermeidet dadurch von vornherein Problematiken, die durch gleichzeitige Änderungen an einer Datei durch veschiedene Personen entstehen können. Locking kann im Umgang mit binären Dateiformaten von Vorteil sein, da

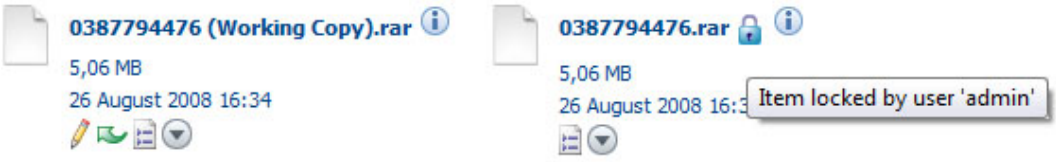

Abbildung 3.1: Check-Out im Kontext eines DMS

ein Merge bei diesen Dateien schwer möglich ist (es wird ab Kapitel 6 noch auf Ansätze eingegangen). Binäre Dateiformate kommen bei der Bearbeitung von CAD-Plänen (AutoCAD DWG-Format) oder auch bei der Textverarbeitung (Microsoft Word 2003 DOC-Format) zum Einsatz.

Security DMS bieten "out-of-the-box"ausgefeiltere Möglichkeiten zur Absicherung der Inhalte: Berechtigungen (ACL, Access Control Lists; z.B. BROWSE, READ, RELATE, VERSION, WRITE, DELETE in Documentum, s. auch [26, Seite 33]), Rollen und Gruppen. Das DMS Alfresco nutzt für Security die mächtige Bibliothek Acegi (aka Spring Security [24]), mit deren Hilfe sich sehr schnell verschiedenste Security Szenarien umsetzen lassen. VCS bieten üblicherweise nur einfache Schreib- und Lesezugriffsberechtigungen (und manche nicht einmal dies, wie in Kapitel 4 gezeigt wird).

Workflow DMS enthalten mächtige Workflow Implementierungen. Alfresco nutzt JBoss JBPM [25], Documentum enthält ebenfalls ein mächtiges Workflow Manager Tool (zu sehen in [26, Seite 27]). VCS erlauben das Ausführen von Scripts zu definierten Zeitpunkten (z.B. nach der Übertragung der Inhalte).

SVN[38] - als ein VCS Vertreter - bietet neun verschiedene Möglichkeiten an, in den Ablauf der Dateiübertragung in das Repository einzugreifen (s. auch [10]). Dies sind post- und pre-Aktionen für Commit, Lock, Unlock und Änderung von SVN-Properties bzw. eine Aktion zu Beginn des Commit. Für eine konkrete Aktion muss ein Skript (in verschiedenen Sprachen möglich) implementiert werden.

Dies ist mit in DMS anzutreffender Workflow Funktionalität nicht vergleichbar. Workflowsysteme erlauben das Zuweisen von Aufgaben zu Benutzergruppen, eine Zeitablaufsteuerung (z.B. falls keine Bearbeitung bis zu einem bestimmten Zeitpunkt erfolgt ist, wird eine Benachrichtigung ausgesandt) und geben mitunter die Umgebung vor (internes System), in der die Aufgabe (Aktivität) durchgeführt werden soll. Anfererseits kann durch das Workflowsystem auch ein externes System eingebunden werden, welches eine Nachricht mit dem Ergebnis übermittelt.

Alfresco[3] - ein DMS - enthält eine Anzahl von eingebauten Regeln, die im Rahmen von Workflows verwendet werden können (s. auch [43]). Ein paar Beispiele hierfür wären: Ausführen eines Scriptes, Bewegen der Datei in ein anderes Verzeichnis, Transformieren eines Dokumentes vom Winword-Format nach PDF. Es ist ein vollwertiges Workflowsystem integriert, um komplexere Abläufe umzusetzen (JBPM[25] - JBoss Business Process Management)

Merging VCS kennen zwei Möglichkeiten um mit gleichzeitigen Änderungen umzugehen: Eines ist die Lock-Modify-Unlock Lösung. Die andere ist die Copy-Modify-Merge Lösung (s. auch [10]). Diese wird nur von VCS unterstützt. Eine interessante Idee wäre es, die Copy-Modify-Merge Lösung in einem DMS mittels Workflow und einer zu implementierenden Bestätigungsmöglichkeit von angezeigten Unterschieden umzusetzen.

Patching Dies ist ebenfalls nur ein Feature von VCS (in manchen Fällen eher nur Teil eines grafischen User Interfaces zum VCS). Es besteht die Möglichkeit einen Patch ([50] zu erstellen. Ein Patch enthält die Unterschiede zweier (üblicherweise Text-)Dateien. Dies wird genutzt, um die Änderungen von einer Datei auf eine andere zu übertragen.

Visualisierung Im Falle von Visualisierung verfolgen VCS und DMS verschiedene Ziele. VCS bieten eher minimale Ansätze bzw. wiederum über grafische Interfaces die Möglichkeit, in Repositories zu stöbern. DMS visualisieren recht aufwändig Verzeichnisstrukturen (Workspaces, Folders, ...) in verschiedenen Ausgabeformaten (im Webbrowser, als Dateisystem, FTP-Server. [43, Seite 115]). Die Visualisierung von Versionsunterschieden fehlt in DMS leider.

Suchen/Navigieren Der Navigationsaspekt hängt stark mit der Visualisierung zusammen. Suchen ist in DMS gut umgesetzt, ist aber in VCS auch zu finden (vor allem in distributed VCS, die die lokale Kopie durchsuchen). In DMS wird üblicherweise auch eine Volltextsuche unterstützt.

Collaboration Collaboration wird nur von DMS unterstützt und kann unterschiedlich stark ausgeprägt sein. Vorstellbar sind Diskussionforen, Einladen von Benutzern (im Grunde ad hoc Berechtigungsvergabe) und Diskussionen zu bestimmten Inhalten (Kommentarabfolgen).

Wie man sieht, bieten VCS und DMS durchaus recht ähnliche Funktionalitäten (s. auch 3.1. Man kann daher abhängig von den Anforderungen mit beiden Systemen Inhalte gut verwalten. Ein Copy-Modify-Merge Ablauf kann allerdings eher nur mit VCS umgesetzt werden. Für die Arbeit mit CAD-Plänen ist aber die Unterstützung dieses Ablaufes wichtig. Das später noch im Detail beschriebene Plugin arbeitet daher im Zusammenspiel mit einem VCS.

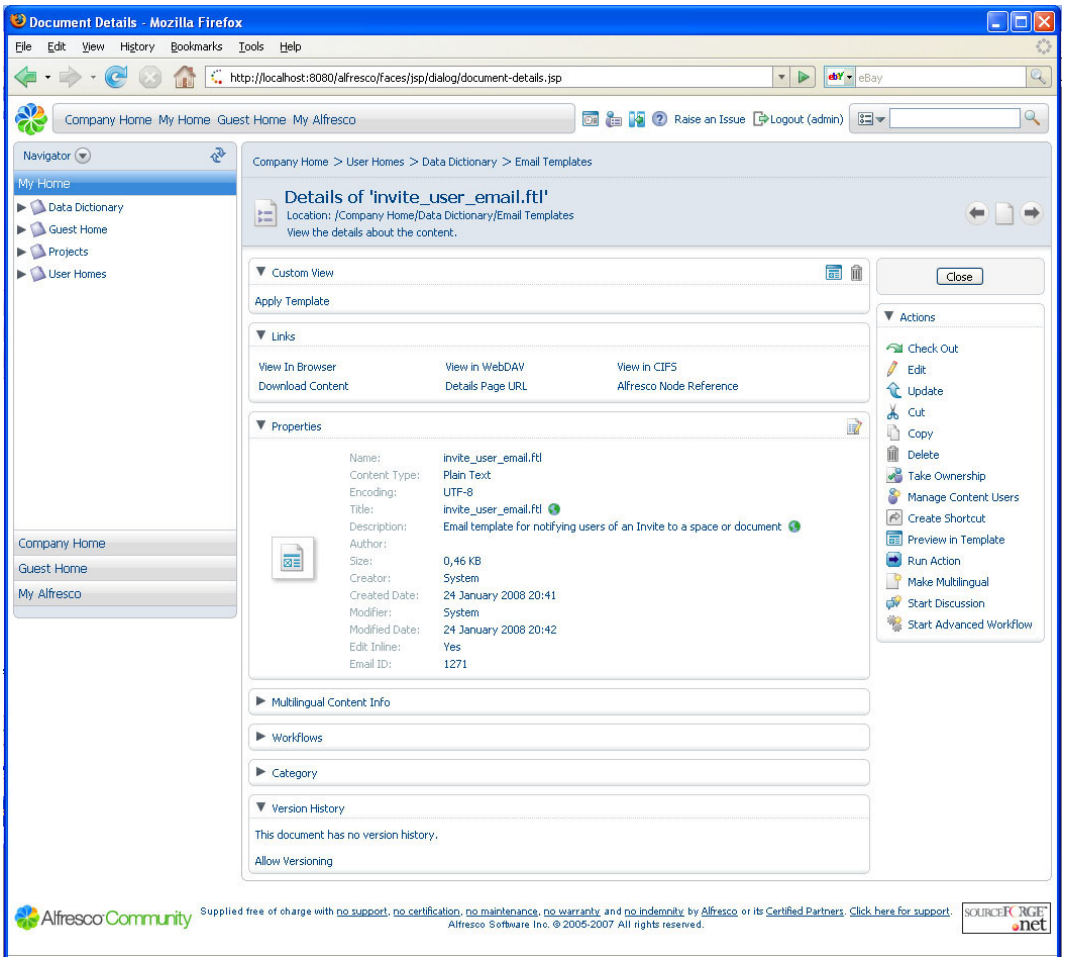

Abbildung 3.2: Visualisierung von Inhalten in Alfresco im Webbrowser

3.4 Software-Engineering, Dokumente und CAD-Plane

3.4.1 Software-Engineering

Im Bereich Software-Engineering sind VCS etabliert. Softwareentwicklung ohne Verwendung von VCS ist undenkbar. Es werden meist textbasierte Dateien verwendet. Diese können z.B. Java-Code darstellen oder XML-Dateien. Entwickler nutzen VCS, um gemeinsam an Softwareprojekten zu arbeiten. Softwareprojekte werden zudem auch oftmals durch räumlich getrennte Teams umgesetzt. VCS verbessern die Nachvollziehbarkeit und Kommunikation zwischen Mitgliedern des Projektes.

Der Ablauf in diesem Bereich ist üblicherweise Copy-Modify-Merge. Falls binäre Dateien involviert sind (z.B. Oracle Forms [54]), sollte der Lock-Modify-Unlock Ablauf angewandt werden.

Im Zuge der Entwicklung wird zuerst aus einem zentralen Repository Co-

de abgeholt (Check-Out), editiert und wieder in das Backend übertragen. Schon hierbei kann es durch Änderungen von verschiedenen Personen an der gleichen Datei zu Konflikten kommen, die manuell aufgelöst werden müssen (mit Hilfe von VCS Funktionalität). Die Situation wird noch komplexer, wenn im Projektverlauf Branches erstellt werden, um neue Features in einem getrennten Bereich zu isolieren. Wenn ein stabiler Zustand erreicht ist, wird der Code aus dem Branch mit dem Hauptzweig wieder vereint. Dies wird Merging genannt. Eine sehr gute Erklärung, welche Strategien für Branching und Merging existieren, findet sich unter [12].

Probleme entstehen üblicherweise beim Merging (diese können durch falsche Branching Strategien verursacht sein). In einfachen Fällen kann das VCS selbst ein Merging durchführen, ohne dass eine Aktion des Benutzers notwendig wäre (eine manuelle Prüfung des Ergebnisses ist trotzdem empfehlenswert). In schwierigeren Merging-Fällen werden die Konflikt verursachenden Dateien gegenüber gestellt und der Benutzer muss einen endgültigen Dateizustand herstellen. Diese Gegenüberstellung von Text-Dateien wird von VCS unterstützt. Der Benutzer muss hierbei aus den betroffenen Dateiversionen für jeden Konflikt entscheiden, welche Inhalte in eine vereinigte dritten Version übertragen werden sollen.

Die in der Softwareentwicklung verwendeten Text-Dateien haben eine dem Programmierer bekannte Syntax und Semantik. Aus diesem Grunde ist Merging in diesem Bereich durch Textvergleich handhabbar.

3.4.2 Dokumente

Ein Dokument (z.B. Diplomabeit, Bericht) wird oft durch eine einzelne Person erstellt. Falls mehrere Personen am Prozess beteiligt sind, hilft eine gute Arbeitsteilung (jede Person kümmert sich um ein bestimmtes Kapitel), um Konflikte durch gleichzeitige Änderungen zu vermeiden. Reviews und Qualitätssicherung führen auch nicht zu Problemsituationen, da diese zeitlich von der Erstellung getrennt sind.

Der Lock-Modify-Unlock Ablauf ist für die Erstellung von Dokumenten üblich und wird durch DMS mittels Check-Out und Check-In Funktionalität unterstützt. Im Falle von Dokumentformaten, die textbasiert sind (nicht binär), kann ein VCS ausreichend sein, wenn man auf erweiterte Funktionen wie Workflowunterstützung verzichten kann. Es kann in diesem Falle auch ein Copy-Modify-Merge Ablauf genutzt werden.

Im Falle von Änderungen an binären Dokumentformaten wird entweder Locking angewandt oder diese müssen in der Applikation selbst (z.B. OpenOffice oder Microsoft Word 2007) verglichen werden. Im Falle von zeitlich hintereinander durchgeführten Änderungen an einem Dokument (das Dokument wird weitergereicht) wird durch die Funktionalität “Änderungsverfolgung”die Vereinigung zu einer Endfassung erleichtert (z.B. in Microsoft Word).

Die Visualisierung von Unterschieden in der natürlichen Umgebung des Dokumentformates unterstützt die kooperative Anstrengung.

3.4.3 CAD-Pläne

CAD-Pläne enthalten komplexe Informationen und können auch im Falle von textbasierten Formaten (z.B. DXF) nicht manuell bearbeitet werden. An CAD-Plänen kann es zu Änderungen durch verschiedene Personen gleichzeitig kommen, da Pläne oft grosse Bereiche abbilden.

Falls der Ablauf für das Bearbeiten des CAD-Dateiformates Lock-Modify-Unlock ist, kann dieser sowohl mit VCS als auch mit DMS umgesetzt werden. Der Einsatz von Locking ist aber mitunter nicht akzeptabel.

Der Copy-Modify-Merge Ablauf kann nur gut in der betreffenden Applikation selbst (z.B: AutoCAD) unterstützt werden. D.h. ein Merging würde ausserhalb von VCS oder DMS abgehandelt werden müssen, wobei ein vom Benutzer geprüftes und vereintes Ergebnis archiviert wird. Dies könnte wiederum mit allen Systemen abgehandelt werden, solange der Mergevorgang selbst separat geregelt wird. Im Endeffekt bedeutet dies, dass im Kontext von VCS und DMS nur Versionierung angewandt wird. Dieser Ablauf ist in Abbildung 1.1 im Detail dargestellt und wird durch das im Rahmen der Diplomarbeit entwickelte Plugin unterstützt.

3.5 Zwei Open Source Beispiele

3.5.1 Open Source DMS Alfresco

Es wurden bereits einige Funktionalitäten von Alfresco[3] im Rahmen des Vergleiches von VCS und DMS vorgestellt. Alfresco wurde unter Verwendung folgender Java-Technologien implementiert: JSF, Spring, Hibernate, jBPM und Lucene. Die Metadatenhaltung ist konform zur Java Content Repository (JCR) Spezifikation. Dies bietet Investitionssicherheit: es kann auf die Daten mit einer definierten API zugegriffen werden. Nicht zuletzt gibt es für eigene Erweiterungen ein eigenes Software Development Kit (SDK).

Versionierung wird in Alfresco auf verschiedene Arten unterstützt:

- Diese kann pro Datei aktiviert werden.
- Sie kann pro Verzeichnis angewendet werden, indem ein sogenannter Versionierungsaspekt im Rahmen einer vordefinierten Workflowregel benutzt wird.
- Versionierung kann auf bestimmte Dateitypen angewendet werden. Dies wird in einer Konfigurationsdatei festgelegt (Content Model).

- Es kann in einer Konfigurationsdatei Versionierung global eingestellt werden.

Ein DMS wie Alfresco kann für kooperatives Arbeiten mit CAD-Plänen durchaus geeignet sein. Die verschiedenen Zugriffsmöglichkeiten, kombiniert mit einem enterprisereifen Security-System, Workflowunterstützung und Collaboration können Alfresco (oder DMS allgemein) interessant für diese Problemstellungen erscheinen lassen.

Alfresco hat interessante Ansätze für kollaboratives Arbeiten im Bereich Web Content Management (WCM). Das Konzept nennt sich "Transparent Layers"und lässt sich gut auf Textdateien anwenden. Ein transparentes Verzeichnis bzw. Datei verhält sich ähnlich wie ein symbolischer Link unter Unix. Nachdem aber Änderungen durchgeführt werden, wird eine private Kopie des Originals bearbeitet.

Verwendet werden die transparenten Layers in der "collaborative content production". Damit sollen Web Content Management Staging Prozesse unterstützt werden (Author, Reviewer, Staging).

Alfresco bietet verzeichnis- und dateibasiertes transparent Layering. Dieses basiert auf einem reifen und integrierten Versionierungssystem (s. [2] und [1].

3.5.2 Beispiel OooSVN

Es wird nun das OooSVN Plugin[21] diskutiert und analysiert. Dieses Plugin kann in OpenOffice[32] mit dem VCS Subversion (SVN) als Repository verwendet werden und illustriert auf einfache Weise einen Ansatz für kooperatives Arbeiten.

Die Installationsumgebung ist wie folgt:

- WindowsXP mit Service Pack 2
- OpenOffice v2.3 mit JDK 1.5.06
- Cygwin[40] v1.5.4-2
- Subversion for Windows v1.4.5
- OOoSVN v0.38

Das OOoSVN Plugin läuft offiziell unter Linux bzw. Mac OS X. Die Test-Installation wurde unter Windows mit Cygwin vorgenommen, um einen Eindruck von der allgemeinen Verwendbarkeit zu gewinnen (OpenOffice läuft auf jedem OS mit einer einigermaßen aktuellen Java Virtual Machine). Die technischen Details zur Inbetriebnahme des Plugins unter Windows und die Herausforderungen dabei sind im Anhang A.1 beschrieben. Die Installation der Distribution (ein ZIP-Archiv) gestaltet sich einfach. Über den Menüeintrag *Tools* → *Extension*

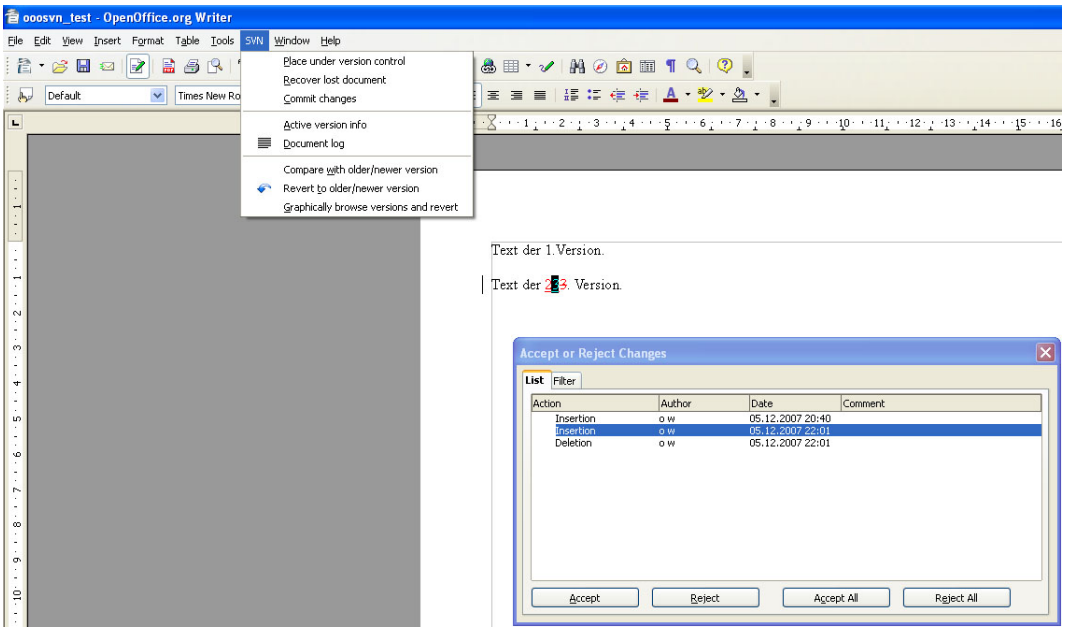

Abbildung 3.3: Visualisierung von Differenzen - Konflikte editieren mit OOoSVN

Manager, lässt sich das Plugin in OpenOffice hinzufügen. Es wird sodann (nach einem Neustart) ein Menüeintrag *SVN* sichtbar.

Das Plugin ist in OpenOffice.org Basic geschrieben und nutzt für die Subversion Anbindung sechs Bash-Shell Skripts. Diese sind auch der Grund für die Verwendung von Cygwin unter Windows. Falls ein anderes VCS (z.B. DARCS) verwendet werden soll, ist dies die Stelle an der angesetzt werden muss. Der Code ist nicht sehr wiederverwendbar geschrieben (s. auch A.1). Die voraussichtlich weiteren Funktionalitäten des Plugin sind ansatzweise in den definierten Dialog-Bildschirmen sichtbar. Diese werden momentan nicht verwendet. Diese sollen die Verwaltung von verschiedenen SVN Repositories ermöglichen.

Damit werden allerdings auch die aktuellen Limits des Plugins sichtbar:

- Es ist nur ein (lokales) Repository verwendbar.

- Es lassen sich Dokumentnamen nur einmal für die Versionierung verwenden, da der Dokumentname direkt in einen Verzeichnisnamen einfliesst. Dies führt auch zu im Anhang beschriebenen Problemen, falls Leer- bzw. Sonderzeichen in Dateinamen verwendet werden.

Die Verwendung eines lokalen (also nicht gemeinsam genutzten) SVN-Repository verhindert natürlich eigentlich kooperatives Arbeiten. Man kann, wie im Anhang beschrieben wird, auch einen URL zu einem nicht lokalen SVN-Repository angeben. Dann kann gemeinsam und verteilt darauf zugegriffen wer-

den (ein gemeinsam genutztes Netzwerklaufwerk wäre gegebenenfalls auch verwendbar). Das oben genannte Limit ist also mit wenig Aufwand zu umgehen.

Das Plugin illustriert allerdings zwei von den oben genannten technischen Ansätzen: *Versionierung* und *Visualisierung von Differenzen*(s. auch Abb. 3.3) Für die Anzeige der Unterschiede wird die native Umgebung von OpenOffice verwendet (Es wird die UNO[39] Funktion CompareDocuments verwendet). Dadurch gestaltet sich das Nachvollziehen von Änderungen effizient. Die Versionierung ermöglicht den Vergleich.

Zusammenfassend lässt sich zum OOoSVN-Plugin sagen, dass die grundlegende Idee, nämlich das Visualisieren von Differenzen verschiedener Versionsstände in der nativen Umgebung einer Datei, gut ist, aber das Plugin momentan noch in keinem Zustand ist, um eine produktive Verwendung empfehlen zu können. Dieser Ansatz wird allerdings in den Kapiteln 6 und 7 im CAD-Kontext weiterverfolgt werden.

Das OooSVN Projekt ist in das Projekt ODF-SVN [22] übergegangen.

Kapitel 4

Vergleich einzelner VCS: CVS, SVN und DARCS, GIT

4.1 Zugrundeliegende Philosophien

Es gibt zwei Arten, Daten in Versionierungssystemen zu halten. Daraus leiten sich weitere Implikationen ab, die in den folgenden Sektionen beschrieben sind.

4.1.1 Client-Server Modell

Beim Client-Server Modell gibt es ein zentrales Repository, in welchem alle Änderungen verfolgt werden können. Benutzer verwenden lokale Kopien. Änderungen an diesen Kopien werden erst für alle verfügbar, wenn diese in das zentrale Repository übetragen worden sind ("Commit").

Wie man leicht schliessen kann, lässt sich Locking nur bei Verwendung des Client-Server Modelles anwenden.

Es kann Umgebungen geben, die ein Client-Server Modell bevorzugen, um eine stärkere Kontrolle ausüben zu können. Dies kann z.B. mit detaillierten Berechtigungen auf Verzeichnisebene (in einem Projekt) unterstützt werden.

4.1.2 Verteiltes Modell

Verteilte Versionskontrolle kennt eigentlich kein zentrales Repository. Jeder Benutzer hat sein unabhängiges, eigenes. Es kann aber sinnvoll sein, ein gemeinsames Repository, in welchem präferiert Änderungen abgelegt werden, zu verwenden.

Der Vorteil eines lokalen Repositories (des verteilten Modells) ist, dass man Änderungen auch offline (ohne Verbindung zum einem zentralen Repository) sichern kann bzw. auch rückgängig machen kann. Dadurch, dass alles lokal vorliegt, sind Vorgänge wie Commit entsprechend schnell.

Ein Nachteil kann der erhöhte Platzbedarf für das lokale Repository sein und der vermeintliche Kontrollverlust. Dieser kann aber durch Verwendung "zentraler"Repositories abgemildert werden. Der Umgang mit binären Dateien kann ohne Locking erschwert sein.

Das verteilte Modell ist besonders für globales Arbeiten (Git wird z.B. zur Weiterentwicklung und Wartung des Linux-Kernels verwendet) geeignet. Skalierbarkeit ist ein positiver Nebeneffekt.

4.1.3 History Modelle

Ein History Modell beschreibt, wie Änderungen an Verzeichnissen und Dateien (einem Verzeichnisbaum) im VCS intern abgelegt werden.

Die verschiedenen History Modelle seien kurz erklärt:

- Ein Changeset History Modell speichert den Verzeichnisbaum vor einer Änderung und ein Changeset mit den Änderungen. Dies ist ein Set von Deltas für jede Datei bzw. Verzeichnis, an der/dem Änderungen vorgenommen worden sind.
- Ein Snapshot History Modell speichert den Verzeichnisbaum vor einer Änderung und nach einer Änderung. Es wird ein gerichteter Graph von Zuständen der betroffenen Elemente (Datei, Verzeichnis) gespeichert, wenn ein Commit durchgeführt worden ist.
- Beim Patch History Modell ist nicht der Verzeichnisbaum das fundamentale Objekt, sondern der Patch[42]. Der Patch ist nicht die Differenz zweier Bäume, sondern ein Baum ist das Resultat von angewandten Patches auf einen leeren Baum.

4.2 Vergleich

CVS und SVN sind Vertreter des Client-Server Repository Modells für VCS. DARCS und Git nutzen das verteilte Repository Modell (s. oben).

Die Arbeitsweise ist grundsätzlich in allen hier beschriebenen VCS sehr ähnlich: Der Client verbindet sich zum Repository und holt sich eine Kopie des Projektes und überträgt Änderungen später wieder in das Repository.

4.2.1 CVS und CVSNT

CVS ist ein Veteran unter den heute noch verwendeten VCS: Der CVS-Code wurde 1986 öffentlich gemacht (für genauere Informationen was CVS ist: siehe [51]). CVS wird nicht mehr aktiv weiterentwickelt. Patches werden aber noch erstellt [46]. CVSNT [28] ist eine Weiterentwicklung von CVS, die bei Bedarf auch kommerziell unterstützt wird. CVS und CVSNT werden hier genannt und

verglichen (in der nächsten Sektion auch mit SVN), da dieses VCS über viele Jahre den Quasi-Standard in diesem Bereich gelbildet hat.

CVSNT verwendet das Changeset History Modell. CVS verwendet das Snapshot History Modell.

Feature	**CVSNT**	**CVS**
Server		
Authentication via Microsoft Active Directory/SSH	Ja	Nein
LockServer für Datei Level locking	Ja	Nein
Effiziente Speicherung binärer Dateien (mittels binärer Deltas)	Ja	Nein
Server-side Default Optionen (cvsrc)	Ja	Nein
UTF-8 (Unicode) Server	Ja	Nein
Unterstützung für i18n Dateinamen	Ja	Nein
Unterstützung für encrypted Authentication via SSL	Ja	Nein
Client		
Smart Merge mittels MergePoint[27]	Ja	Nein
"Import-and-go"(Kein Purge und Checkout nötig)	Ja	Nein

Tabelle 4.1: Auszug: Vergleich CVSNT und CVS (von [28])

Die Tabelle 4.1 streicht die Features von CVSNT heraus. Diese zeigt damit allerdings eher, welche Defizienzen CVS hatte. Ein Vergleich von CVS und SVN (s. Sektion 4.2.2) zeigt dann weitere Problematiken von CVS auf (die allerdings CVSNT teilweise nicht betreffen; s. auch [52]).

Zusammenfassend kann man zu CVS sagen, dass es mittlerweile genügend (Open Source) Alternativen gibt, um Softwareentwicklung zu betreiben bzw. versionierte Dokumente abzulegen. Eine beliebte Alternative ist SVN, welches im Open Source Bereich und auch im Enterprise Bereich in den letzten Jahren vermehrt eingesetzt wird (s. z.B. [19]).

4.2.2 SVN und CVS

Subversion (SVN) ist ein VCS, welches als Ersatz für CVS gedacht ist ([38] bzw. etwas ausführlichere Ausführungen unter [35])).

SVN verwendet sowohl das Changeset als auch das Snapshot History Modell.

SVN weist folgende Vorteile gegenüber CVS auf:

- Atomic Commits: CVS garantiert nur Atomizität auf Dateiebene (d.h. es wird eine Datei komplett oder gar nicht in das Repository übertragen). SVN bietet dies auf Commit-Level an (also per Changeset).

- SVN unterstützt im Gegensatz zu CVS Versionierung von Verzeichnissen und Metadaten sowie Umbenennungen von Dateien. Mit CVS muss eine

Datei entfernt werden und dann unter dem neuen Namen in das Repository übertragen werden.

- Branching und Tagging sind in SVN als Copy Operation ausgeführt. In CVS wird jedes einzelne File bei diesem Prozess angerührt. Operationen (diff, checkout) auf Branches in CVS sind komplexer und deswegen langsamer.

- SVN benutzt den xdelta [31] Algorithmus, um Unterschiede in binären Dateien zu finden. Dieser ist auch für Text-Dateien gut geeignet. CVS hat mit binären Dateien Probleme: Diese müssen beim Übertragen in das Repository speziell gekennzeichnet werden, da es sonst zu unerwünschten Effekten kommen kann (s. [18]).

- CVS schickt Dateien zum Server (da Speicherplatz am lokalen Rechner einst Mangelware war), um Differenzen zwischen lokaler Kopie und der Version im Repository zu finden. Der Vergleich wird am Server durchgeführt und das Ergebnis zurück an den Client geschickt. Ähnliches passiert im Falle von Updates, Commits und Merges. SVN hingegen hält im Unterverzeichnis .svn/text-base eine Kopie der letzten ausgecheckten Version mit welcher lokal verglichen wird. Unterschiede (diffs) werden im Falle von SVN auch vom Client zum Server gesandt.

SVN bietet mit dem SVNKIT [44] eine gute Integration mit Java an. Diese Bibliothek wird von einigen der Java-IDE Herstellern in ihrer Software verwendet (z.B. JetBrains, Oracle, Borland). Für C# bietet sich SharpSVN [14] an.

Subversion (SVN, [38]) führt ab Version 1.5 diverse Features in der Roadmap, die das Client-server Modell erweitern, wie:

- Merge tracking (für Version 1.5)

- als Langzeitziel: hybrid distributed/centralized version control model

4.2.3 DARCS und CVS

DARCS verwendet ein Patch History Modell[52]. Roundy nennt DARCS allerdings "changeset-oriented, not file-oriented"[41]. DARCS nutzt das verteilte Repository Modell.

- Lokale Änderungen können rückgängig ("unrecord a change") gemacht werden (solange sie nicht publiziert wurden), sogar wenn es nicht die letzte Änderung gewesen ist. Danach können die Änderungen anders festgeschrieben werden. Dies kann zum Beispiel nötig sein, wenn man etwas als ein Set übertragen hat, das besser in zwei Sets abgelegt worden wäre. Oder ein Set ist ohne eine bestimmte Datei, die bei einem Commit vergessen wurde,

nicht komplett. Man fügt eine solche Datei nachträglich dem Set hinzu und erzeugt damit eine saubere Historie (s. auch [15]).

- Das Verschieben von Dateien und Verzeichnissen wird korrekt gehandhabt.
- Token replace: Mittels des Kommandos *darcs replace* können bestimmte konfigurierte Token ersetzt werden. Dies hat den Vorteil, dass die alten Werte (vor dem Ersetzen) beim Merging ersetzt werden. Dies kann z.B. bei Änderungen an Variablen oder Funktionen, die oft in einem Projekt vorkommen.
- DARCS tut sich hingegen mit Merging "schwer". Im Bereich Merging wurde DARCS in der Vergangenheit wegen der Performance kritisiert. Auch in der aktuellen Version 2.0 sind nicht alle Probleme gelöst (s. [53]).
- DARCS bietet keine effiziente Datenhaltung für Binärdateien an. Diese werden immer komplett im Repository abgelegt (nicht als deltas).

CVS	**DARCS**
cvs checkout	darcs get
cvs update	darcs pull
cvs -n update	darcs pull –dry-run (summarize remote changes)
cvs -n update	darcs whatsnew –summary (summarize local changes)
cvs -n update — grep '?'	darcs whatsnew -ls — grep ^a (list potential files to add)
rm foo.txt; cvs update foo.txt	darcs revert foo.txt (revert to foo.txt from repo)
cvs diff	darcs whatsnew (if checking local changes)
cvs diff	darcs diff (if checking recorded changes)
cvs commit	darcs record (if committing locally)
cvs commit	darcs tag (if marking a version for later use)
cvs commit	darcs push or darcs send (if committing remotely)
cvs diff — mail	darcs send
cvs add	darcs add
cvs tag -b	darcs get
cvs tag	darcs tag

Tabelle 4.2: Vergleich von Kommandos in CVS und DARCS

Die Tabelle 4.2 vergleicht CVS und DARCS Kommandos. Beachtenswert sind die Variationen der Kommandos in DARCS, welche das lokale bzw. ein remote Repository abfragen. Auf DARCS wird im folgenden Kapitel noch genauer eingegangen.

4.2.4 Git und SVN

Git[8] wurde 2005 für die Wartung des Linux Kernels geschrieben. Es nimmt für sich in Anspruch schnell zu sein (dies wird z.B. von [45], Kapitel 1.6.2 bestätigt).

Git stellt über 100+ einzelne Kommandos zur Verfügung, es hat daher den Ruf, schwer erlernbar zu sein. Git benötigt ausserdem gelegentliches Packen der Metadaten, da ansonsten die Performance sinkt und der Platzverbrauch stark zunimmt.

Git verwendet das Snapshot History Modell und nutzt wie auch DARCS ein verteiltes Repository Modell.

Nennenswerte Features sind:

- Git unterstützt Branching und Merging besonders gut (sprich: es ist schnell).
- Git bietet kryptografische Sicherheit der History. Damit wird nachträgliches Manipulieren der Geschichte verhindert (dies macht bei einem Projekt wie dem Linux Kernel Sinn).
- Das Repository kann (muss aber nicht) gesäubert werden (dies ist bei SVN z.B. nicht möglich).

Um Merging besser zu unterstützen, verwendet Git keine “einfachen ”Revisionsnummern verwendet (s. [52]), sondern z.B. SHA-1 Hashes. Damit lassen sich Revisionen eindeutig(er) identifizieren und verfolgen. Falls tatsächlich eine Revision ausgezeichnet werden soll, wird ein Tag vergeben (wie auch im Falle von Client-Server Modellen).

Git bietet gute Möglichkeiten, weiterhin auch gegen ein zentrales SVN Repository zu arbeiten (s. [57]). Dieses Kommando wird allerdings durch die Windows Version von Git (Version 1.5.4) noch nicht unterstützt.

In Tabelle 4.3 werden gängige Kommandos von Git und SVN gegenüber gestellt.

Git wird von einer stattlichen Anzahl von Projekten, die hauptsächlich mit Linux zusammenhängen, verwendet [34].

	GIT	**SVN**
Schnellstart		
Projekt lokal kopieren	git clone url git pull	svn checkout url svn update
Commiting		
Setup lokales Repository	git init git add . git commit	svnadmin create *repo* svn import *file://repo*
Geänderte Dateien auflisten	git diff	svn diff — less
Geänderte Dateien ab Revision	git diff *rev path*	svn diff -r*rev path*
Änderungen applizieren	git apply	patch -p0
Kurzliste geänderter Dateien	git status	svn status
Zurück zur letzten Revision	git checkout *path*	svn revert *path*
Add, Move, Remove	git [add, mv, rm] *file*	svn [add, mv, rm] *file*
Änderungen übertragen	git commit -a	svn commit
Browsing		
History überprüfen bzw.	git log git blame *file*	svn log — less svn blame *file*
Datei, Verzeichnis oder Commit anzeigen	git show *rev:filepath* git show *rev:dirpath* git show *rev*	svn cat *url* svn list *url* svn log -r*rev url* svn diff -c*rev url*
Tagging und Branching		
Tag erstellen	git tag -a *name*	svn copy *url*/trunk *url*/tags/*name*
Tags auflisten	git tag -l	svn list *url*/tags/
Branch erstellen	git branch *branch*	svn copy *url*/trunk *url*/branches/*branch*
Wechseln zu Branch	git checkout *branch*	svn switch *url*/branches/*branch*
Merging		
Merge HEAD, im trunk stehend	git merge *branch*	svn merge -r *rev*:HEAD *url*/branches/*branch*
Merge best. commit	git cherry-pick *rev*	svn merge -c *rev url*
Änderungen publizieren		
Senden an entfernte Repositories	git push *remote*	-
Patch per Mail senden	git format-patch git send-email	-

Tabelle 4.3: Vergleich Kommandos Git und SVN (s.[7])

Kapitel 5

DARCS im Detail

5.1 Warum Darcs?

DARCS unterstützt textbasierte Formate gut und hat ein einfaches Set an Kommandos (zumindest verglichen mit Git). Einer der grössten Vorteile von DARCS ist die Möglichkeit, vorangegangene Patches (= Unterschiede zwischen Datei- bzw Verzeichnisversionen) nachträglich zu ändern (s. Beispiel auch unter [33]). Mit Patches werden die verteilten Repositories abgeglichen. Dies kann attraktiv für Softwareentwicklung sein und wird hier deswegen genauer vorgestellt.

Es sei ein Arbeitsablauf zur Implementierung dreier Features gezeigt:

- Ein Checkout von einer bestimmten Code-Basis wird vorgenommen.
- Feature X wird implementiert.
- Commit.
- Ein von X unabhängiges Feature Y wird implementiert.
- Commit.
- Ein von X und Y unabhängiges Feature Z wird implementiert.
- Commit.
- Übertragen des Ergebnisses in die Code-Basis.

Nun realisiert der Entwickler in obigem Ablauf, dass in Feature X eine Korrektur anzubringen ist. Damit bei der Übertragung der Ergebnisse vom lokalen Repository zur Code-Basis (darcs push bzw. darcs send) der Patch für Feature X nicht als Flickwerk von kleinen Patches, die gemeinsam Feature X repräsentieren, erscheint, muss der originale Patch für Feature X angepasst werden.

In DARCS wird dieser Anwendungsfall unterstützt, da die Patch-Theorie den Patch für Feature X unabhängig von den anderen (unabhängigen) Patches handhaben lässt.

5.2 Vorstellung Patch-Theorie

Die folgenden Ausführungen sind zumeist von [42], Kapitel "theory of patches"übernommen. Diese erklären die treibende Idee hinter DARCS: Die Patch-Theorie.

5.2.1 Einführung

Ein Patch stellt eine Änderung an Verzeichnissen bzw. Dateien dar. Es gibt entweder einfache (z.B: das Hinzufügen/Löschen einer Datei, das Umbenennen eines Verzeichnisses oder das Ersetzen eines Bereiches in einer Datei) oder composite Patches, welche viele der gennanten einfachen Änderungen umfassen. Die Patch-Theorie ist von den betroffenen Daten unabhängig.

Die definierende Eigenschaft eines Patches ist, dass dieser auf einen Verzeichnisbaum angewendet werden kann, und dadurch eine Änderung bewirkt. Ein Patch wird durch eine Repräsentation bestimmt und einen Satz von Regeln, die das Verhalten im Zusammenhang mit einem Patch-Typ festlegen. Die Repräsentation eines Patches definiert, welche Änderung ein Patch durchführt und muss im Kontext eines spezifischen Baumes definiert sein. Der einfachste Weg, um einen Baum zu definieren, ist das Anwenden von Patches auf eine leere Struktur. Der Kontext eines Patches bestimmt sich aus dem Set von Patches, die diesem vorangehen.

5.2.2 Beziehungen zwischen Patches

Patches stehen in DARCS entweder in sequentieller oder paralleler Beziehung. Im Falle sequentieller Patches werden diese einfach nacheinander angebracht und bilden eine Komposition, die wiederum ein Patch ist.

Parallele Patches haben einen identischen Kontext (d.h. ihre Repäsentation wird auf identische Bäume angewandt). Falls Patches in keine der beiden genannten Kategorien fallen, müssen diese so manipuliert werden, dass sie entweder sequentiell oder parallel angewandt werden können.

Die fundamentalste und einfachste Eigenschaft von Patches ist deren Invertierbarkeit.

> Das Inverse von P ist P^{-1}. Eine Komposition $P^{-1}P$ führt zu keinen Änderungen am Verzeichnisbaum.

Das Inverse einer Sequenz von Patches ist die Sequenz der Inversen der Patches in umgekehrter Reihenfolge: $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$.

Die Anwendung eines Patches und des Inversen führt zu keiner Änderunng am Baum.

5.2.3 Reihenfolgenänderungen bei Patches

Patchkompositionen bestehen aus einer Serie von Patches, die sequentiell angewandt werden sollten. Ein Weg, um den Kontext eines Patches zu ändern ist durch Kommutation (also Reihenfolgenänderung). Diese Operation kann scheitern, da z.B: ein Löschen eines Inhaltes vor einem Einfügen desselben nicht das gleiche Ergebnis haben kann.

5.2.4 Merge von Patches

Ein zweiter Weg, um den Kontext von Patches zu ändern, ist durch eine Merge Operation. Hierbei werden zwei unabhängige (parallele) Patches in einen sequentiellen Zusammenhang gestellt. Das Ergebnis (eine Sequenz zweier Patches) muss kommutieren können.

Die Durchführung des Merging berücksichtigt mehrere Fälle. Einer setzt voraus, dass es keine Konflikte gibt, geht aber auch nicht von einer einfachen Konstellation aus. D.h. es werden beispielsweise in einer Datei Zeilen eingefügt, die eine Umnumerierung für das Anbringen weiterer Änderungen bedeutet, die sich in der Datei hinter den neuen Zeilen befinden.

Konflikte beim Merging werden durch einen speziellen Patchtyp unterstützt, der "Merger"genannt wird. Dieser Patch enthält die konfliktbehafteten Patches. Im Falle, dass Zeilen in einer Datei das Problem auslösen, wird eine Lösung wie bei CVS angewandt: es werden beide Versionen in die Datei eingefügt.

DARCS versucht für Merger Patchtypen die gesamte Sequenz wiederherszustellen, die zu dem Problemzustand geführt hat. Es werden hierbei alle involvierten Patches ermittelt.

5.2.5 Anwenden von Patches

Es gibt sogenannte "Hunk Patches"und "Token Replace Patches".

Hunk Patches sind komplexe Datei-Patches. Hierbei werden Sets von Zeilen durch Sets anderer Zeilen ersetzt. Jedes dieser Sets kann leer sein und entspricht dann entweder einem Hinzufügen oder Löschen von Zeilen.

Token Replace Patches ersetzen alle Instanzen eines Tokens, welches als einfacher regulärer Ausdruck formuliert sein muss (also z.B. $[A-Za-z_0-9]$ im Falle eines Variablennamens). Diese Art von Patch führt üblicherweise nicht zu einem

Konflikt, ausser wenn zufällig die gleiche Ersetzung in einem Patch vorgenommen wird.

5.3 Limits

DARCS in der Version 1.x konnte in der Handhabung problematisch sein, falls der sogenannte *big conflict bug* auftrat. Auch DARCS in der Version 2.x löst das Problem (big conflict bug aka exponential merge problem) nicht vollständig (s. [53]). Verschachtelte Konflikte führen zu einem Problem beim Merging. D.h. je mehr Patches auf einen konfliktbehafteten Patch existieren, desto länger die weiter oben beschriebene Ermittlung der involvierten Patchsequenz.

Ein Resolution für einen Konflikt der beschriebenen Art muss auf alle Branches angewandt werden, da ansonsten zukünftige Weiterentwicklungen in den Branches zu Konflikten zur Resolution führen und die Verschachtelung nur tiefer wird!

Andere Lösungsansätze für das genannte Problem wären das Mergen von Hand ausserhalb von DARCS.

5.4 Anwendbarkeit für kollaboratives Arbeiten

DARCS lässt sich mit textbasierten Formaten nach ein wenig Einarbeitung verwenden. Ein grafisches User Interface von der Qualität wie es CVS, SVN und andere VCS bieten, fehlt momentan leider. Dies fällt gelegentlichen Benutzern von DARCS unangenehm auf. Die Handhabung von binären Formaten oder auch schon von XML Formaten ist ohne grafische Unterstützung oder Einbettung in die natürliche Applikation des Dateiformates nicht effizient möglich.

Der Patch Ansatz hat durchaus seinen Charme, allerdings scheinen sich die VCS SVN, Git und Mercurial [45] aktiver weiter zu entwickeln (d.h. neue Features und Fehlerbehebungen finden schneller Eingang).

Kapitel 6

Architektur und Design: CAD-Plugin Konfliktresolution

6.1 Vision

CAD Pläne werden oftmals durch mehrere Personen überarbeitet. Unter Umständen kommt es auch zu Konflikten nach gleichzeitigen Änderungen. Bei einem Einsatz von VCS können diese erkannt werden. Allerdings müssen Änderungen gesichtet und im Falle von Konflikten aufgelöst werden. Dies kann eine zeitaufwändige und mühselige Tätigkeit sein.

Ein im Rahmen der Diplomarbeit erstelltes Werkzeug zur Unterstützung der genannten Arbeitsabläufe mit Anbindung an VCS ("CADComparator") wird im folgenden beschrieben.

In der ersten Version unterstützt das Werkzeug die Visualisierung von Differenzen und das Auflösen von Konflikten eingeschränkt. Die Einschränkungen ergeben sich teilweise durch Umstände ausserhalb des Einflussbereichs des Autors. Auf diese wird in Kapitel 7 detailliert eingegangen.

Weitere Einsatzmöglichkeiten für das Plugin wären:

- Erstellen von maschinell verarbeitbaren Differenzlisten für Fremdsysteme.
- Unterstützung von CAD Plan Management durch Anbindung an eine relationale Datenbank.
- Einsatz im Rahmen von Qualitätssicherung von CAD-Plänen. Dies könnte durch Erweiterung um Regeln geschehen, denen ein Plan genügen muss.

6.2 Anforderungen an das Plugin

Es soll eine einfache Möglichkeit geschaffen werden, mit der Konflikte und Änderungen an CAD Plänen (dies entspricht im VCS Sprachschatz "merge"und "diff") eingesehen und gegebenenfalls aufgelöst werden können. Ein solches Tool soll nach Möglichkeit lizenzfrei und effizient handhabbar sein. Um kooperatives Arbeiten zu unterstützen, soll auch eine Anbindung an VCS gegeben sein.

Der Vorgang, der hierbei abläuft, sieht aus wie in Abbildung 1.1 dargestellt. Dieser Prozess wird durch ein VCS und durch das erarbeitete Tool unterstützt.

Folgende nicht-funktionale Anforderungen werden besonders berücksichtigt:

- Aussehen und Handhabung
- Benutzbarkeit
- Portierbarkeit und Übertragbarkeit
- Leistung und Effizienz
- Wartbarkeit und Änderbarkeit

Das Ergebnis ist in Abbildung 6.1 zu sehen.

6.3 Architektur

6.3.1 Grundlegende Entscheidungen

Das Einlesen und Schreiben von CAD-Dateiformaten (DWG, DXF, ...) ist keine einfache Aufgabe. Die Darstellung der CAD-Pläne selbst ist ebenfalls aufwändig. Es wurde aus diesen Gründen eine Bibliothek gesucht, welche diese Funktionen bereits beherrscht. Es sollte ausserdem keine Installation weiterer CAD-Software notwendig sein.

Daher wurden folgende Entscheidungen getroffen:

- Die grundlegenden CAD Funktionen (Import verschiedenster Formate - unter anderem DWG und DXF; grafische Darstellung) werden von der Bibliothek CAD Import .Net in der Version 6.3 zur Verfügung gestellt [29].
- CADComparator ist daher in C# auf Basis des .NET Framework 2.0 erstellt worden. D.h. zur Ausführung wird das .Net Framework in der Version 2.0+ benötigt.
- Die Anbindung des Plugins an ein VCS wird mittels TortoiseSVN [13] realisiert. TortoiseSVN ist frei von Lizenzkosten und für die Windows-Platform verfügbar. Die Entscheidung dafür ist in der eleganten Einbindung begründet (s. Abbildung 6.8). Es könnten andere VCS genauso verwendet werden.

Abbildung 6.1: CADComparator in Aktion

6.3.2 Schnittstellen

- Das Zusammenspiel mit einem VCS wird über eine Command-Line Schnittstelle realisiert. Damit ist das Plugin unabhängig von einem bestimmten VCS. Das Plugin vergleicht CAD-Dateien, die über diese Schnittstelle benannt werden. Die Dateien können z.B. im Dateiformat DWG oder DXF vorliegen.

- Die Ergebnisse eines Vergleiches können als Schema constrained XML gespeichert werden.

- Eine mit CADComparator editierte CAD Datei (Merge-Ergebnis) kann grundsätzlich als DXF Datei gespeichert werden (nicht jedoch als DWG, da dieses Ausgabeformat von der verwendeten Bibliothek nicht unterstützt wird). Es existieren aber Tools, die eine solche Formatumwandlung unterstützen (z.B: AnyDWG [5]).

6.3.3 Umsetzung nicht-funktionaler Anforderungen

- Eine grafische Oberfläche, die dem üblichen Aussehen unter Windows entspricht, sorgt für ein vertrautes Handhabungsgefühl.

- Der Benutzer wird gut bei der Hauptaufgabe - dem Überprüfen von Unterschieden und Übertragen von Änderungen - unterstützt. Durch die Integrationsmöglichkeit mit einem VCS Werkzeug (getestet mit TortoiseSVN) wird sofort die Umgebung für effizientes Vergleichen hergestellt.

- Die einfache Installierbarkeit und Übertragbarkeit wird durch die kompakte Auslieferung angesprochen. Es entstehen im Zielverzeichnis der Installation drei Dateien (ein Executable und zwei dynamic Link Libraries), die das komplette Software-Paket darstellen. Dies bedeutet, dass diese Dateien für den Betrieb der Software ausreichen (es werden keine Registry Keys geschrieben oder ähnliche Einstellungen vorgenommen).

- Leistung und Effizienz werden durch aufwändigeren Programmcode unterstützt. So werden gewisse Listen nicht komplett aufgefrischt, sondern zu den richtigen Zeitpunkten inkrementell aktualisiert (z.B. Baumdarstellung).

- Wartbarkeit und Änderbarkeit wird durch entsprechendes Design (z.B. Package Struktur), Anwendung objektorientierter Prinzipien und Refactoring erreicht.

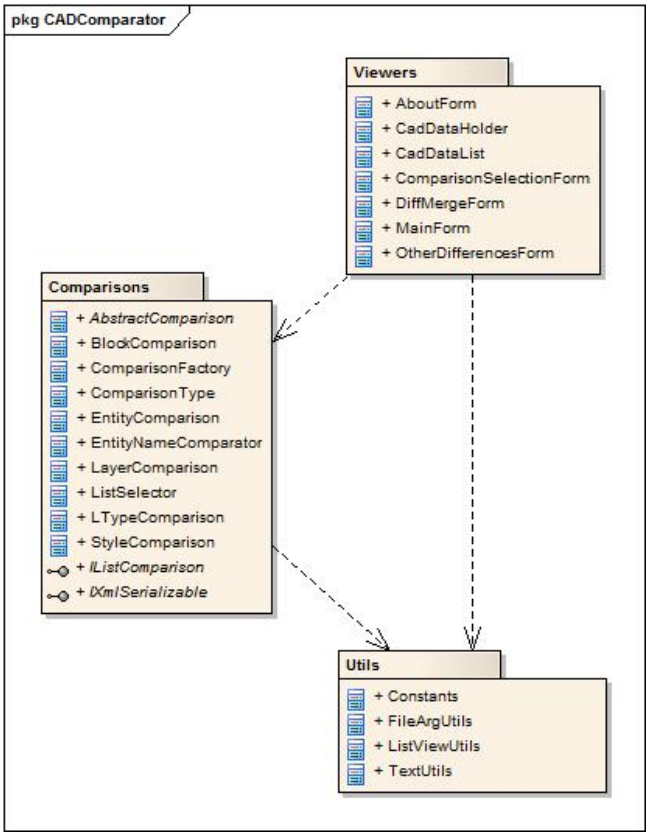

Abbildung 6.2: Package Struktur CADComparator

6.4 Design

Das Programm basiert auf dem Programm "EditorDemo", welches der Bibliothek CAD Import .Net beiliegt. Dieses wurde jedoch massiv erweitert und komplett restrukturiert. Die Bibliothek bietet einfachen Zugriff auf CAD-Objekte (z.B. Entitäten wie Linien, Texte, Kreise, ...) und unterstützt die Visualisierung derselben umfassend.

Das Programm kann diverse CAD-Dateiformate einlesen und stellt die Objekte eines Planes sauber typisiert (es existiert eine Klassenhierarchie in der Bibliothek) über diverse Methoden zur Verfügung. Es gibt also keine Einschränkungen in der Weiterverarbeitung der Pläne (z.B. Vergleiche, Auflistung der enthaltenen Objekte, ...), wenn das Format durch die Bibliothek gelesen werden kann.

Die Paketstruktur der Applikation ist in Abbildung 6.2 zu sehen. Das Package *Utils* stellt unterstützende Klassen zur Verfügung. Die Inhalte der Packages *Comparisons* und *Viewers* sind detaillierter in Abbildungen 6.3 und 6.4 gezeigt.

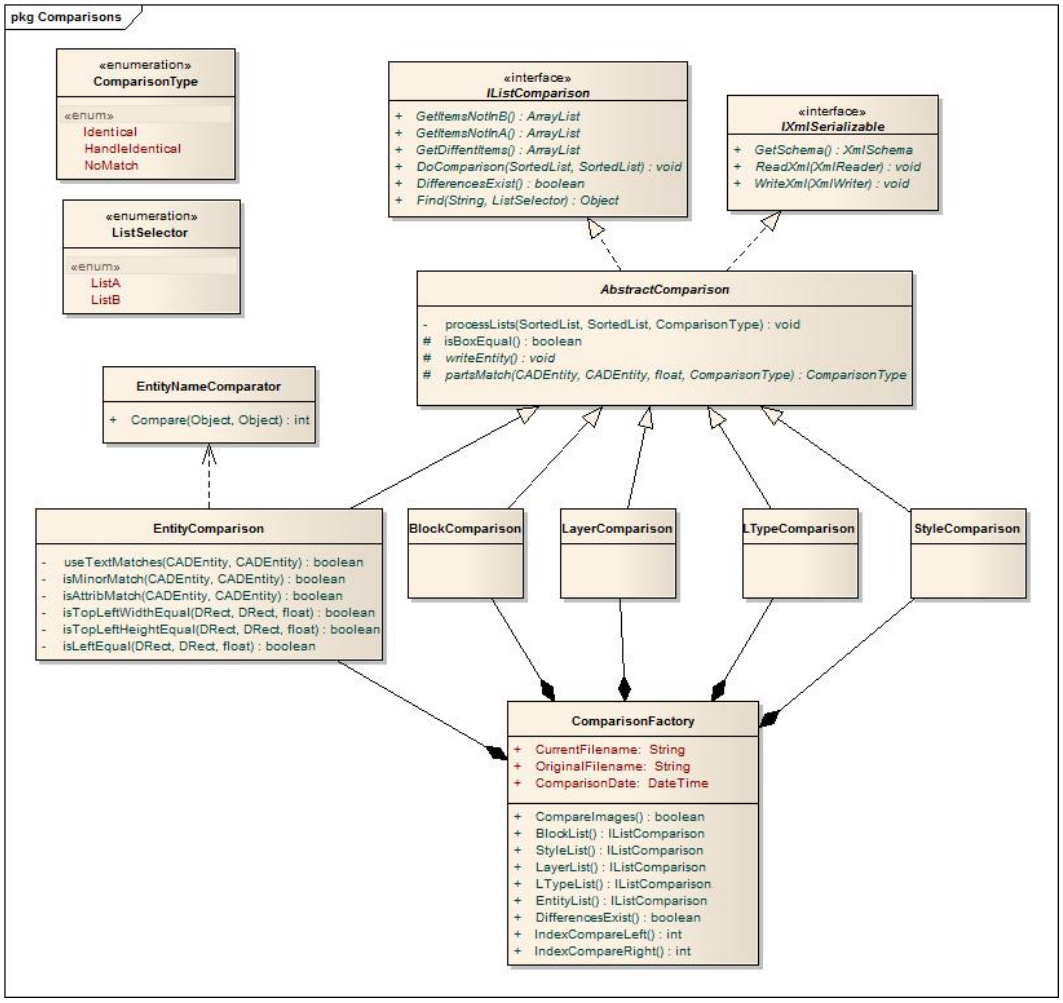

Abbildung 6.3: Package Inhalt Comparisons

6.4.1 Comparisons Package Details

Um Unterschiede zwischen zwei CAD-Plänen zu finden, werden die Elemente eines CAD-Planes nach Typen eingeteilt. Dies sind zum Beispiel Zeichenstile (Styles), Layer, Blöcke und Linetypes (LType). Die Typen werden in zwei Listen abgelegt (die jeweils einem Plan zugeordnet sind). Verglichen werden daher immer zwei Listen (in der Folge Liste A und Liste B genannt). Dies ist durch die Enumeration *ListSelector* abgebildet.

Es wird zwischen drei Arten von Vergleichen unterschieden (Enumeration *ComparisonType*):

- Identical: Die verglichenen Elemente werden aufgrund bestimmter Attribute als identisch angesehen.
- HandleIdentical: Die verglichenen Elemente sind nicht identisch, werden

aber aufgrund erweiterter Prüfungen als ursprünglich identisch angesehen. Diese Elemente erscheinen in der Differenzliste.

- NoMatch: Die betrachteten Elemente weisen keine definitiven Ähnlichkeiten auf und werden als unterschiedlich klassifiziert. Das Element erscheint damit in einer der beiden Listen, die Insert- bzw. Delete-Operationen abbilden.

Das Interface *IListComparison* gibt einen einheitlichen Zugriff auf die Listenoperation und die ermittelten Differenzlisten vor. Elemente aus den zu vergleichenden Listen kommen entweder in genau einer der Differenzlisten vor oder in gar keiner, falls eine Identität vorliegt:

- *DoComparison(...)* führt den Vergleichsalgorithmus für die übergebenen Listen aus.

- *GetItemsNotInA()* gibt Elemente zurück, die in Liste B hinzugefügt worden sind. Dies kann je nach Sicht bedeuten, dass entweder ein Element in Liste A gelöscht worden ist oder in Liste B tatsächlich ein neues Element hinzugefügt worden ist.

- *GetItemsNotInB()* ist der analoge Fall zu *GetItemsNotInA()*.

- *GetDiffentItems()* enthält die Elemente, die als geändert klassifiziert wurden (Enumeration *ComparisonType.HandleIdentical*).

- *DifferencesExist()* ist eine Methode, die überprüft, ob eine der drei Differenzenlisten Elemente enthält. Damit kann leicht festgestellt werden, ob es Unterschiede gibt.

- *Find(...)* bietet eine einfache Möglichkeit, ein Element aus den Differenzlisten zu beziehen.

Die abstrakte Klasse *AbstractComparison* implementiert die beiden Interfaces *IListComparison* und *IXmlSerializable*. *IXmlSerializable* wird für die XML-Export Schnittstelle benötigt (die Alternative, die Klassen einfach mit Attributierungen zu versehen - wie z.B. auch bei der Klasse *ComparisonFactory* - war hier nicht anwendbar, da die betroffenen Klassen von der CAD Import .Net Bibliothek kontrolliert werden). Der Aufwand für die XML Serialisierung hält sich aber in Grenzen, da nur eine Richtung unterstützt wird. Es ist also nur *WriteXml* zu implementieren. Die abstrakte Klasse gibt hier den Rahmen vor: Von *AbstractComparison* ableitende Klassen müssen nur ein Fragment umsetzen, nämlich die Methode *writeEntity()*.

Die private Methode *processLists(...)* enthält den Vergleichsalgorithmus, die protected Methode *isBoxEqual(...)* ist eine Hilfsmethode und unterstützt die Vergleiche; *partsMatch(...)* ist von den jeweiligen konkreten Klassen zu implementieren. Auf die Details des Vergleichsalgorithmus wird im nächsten Kapitel eingegangen 7.1.1.

Die Klasse *EntityComparison* enthält diverse private Hilfsmethoden zur Kategorisierung von Vergleichen und implementiert die vorgegebenen Methoden *partsMatch(...)* und *writeEntity()*. Die Klasse *EntityNameComparator* wird ebenfalls in nächsten Kapitel im Zusammenhang mit der Behandlung von CADTextelementen 7.1.3 genauer erklärt.

Die weiteren konkreten Klassen *BlockComparison*, *LayerComparison*, *LTypeComparison* und *StyleComparison* unterstützen Vergleiche für ihre jeweiligen Spezialbereiche.

ComparisonFactory koordiniert alle Vergleiche und kapselt die Ergebnisse an einer Stelle. Die Methode *CompareImages()* führt die Vergleiche für alle unterstützten Objekttypen durch. *DifferencesExist()* ist ein einfacher Indikator für das Gesamtergebnis. Die *...List()* Methoden geben Zugriff auf die einzelnen Vergleichsergebnisse. *IndexCompareRight* und *IndexCompareLeft* registrieren den Zusammenhang zu den TabStrips der visuellen Darstellung (es gibt je nach Modus bis zu drei TabStrips in Hauptfenster der Applikation zu sehen: Original, Current und Theirs). Die *public* Attribute der Klasse sind nur als ergänzende Informationen für die XML Export-Schnittstelle gedacht.

6.4.2 Viewers Package Details

Die Klasse *MainForm* (s. auch Abbildung 6.4) ist der Startpunkt für die Applikation und kümmert sich um die visuellen Standardaspekte einer Windows-Applikation (Symbolleisten, Menüpunkte, ...). Hervorzustreichen ist hierbei die in Abbildung 6.5 dargestellte Bildschirmeinteilung. Die Klasse *CadDataHolder* verwaltet die drei markierten Bereiche **Zeichenbereich**, **TreeControl** (links oben) und **PropertyGrid** (links unten). Es ist zu einem bestimmten Zeitpunkt jeweils nur genau ein *CadDataHolder* aktiv und wird von der *MainForm* als *currentImage* angesprochen. Das aktive Element wird durch das **TabControl** (über dem TreeControl) ausgewählt. Alle bekannten *CadDataHolder* sind in der *CadDataList* registriert.

Beim Wechseln des Karteireiters wird ein anderes *CadDataHolder* Element dem *currentImage* in der *MainForm* zugeordnet. Der Index des gewählten Karteireiters entspricht hierbei dem Index der *CadDataList* (die Methode *Get(index)* gibt ein *CadDataHolder* zurück).

Die Applikation kennt zwei Modi:

- den Diff-Modus mit zwei Karteireitern

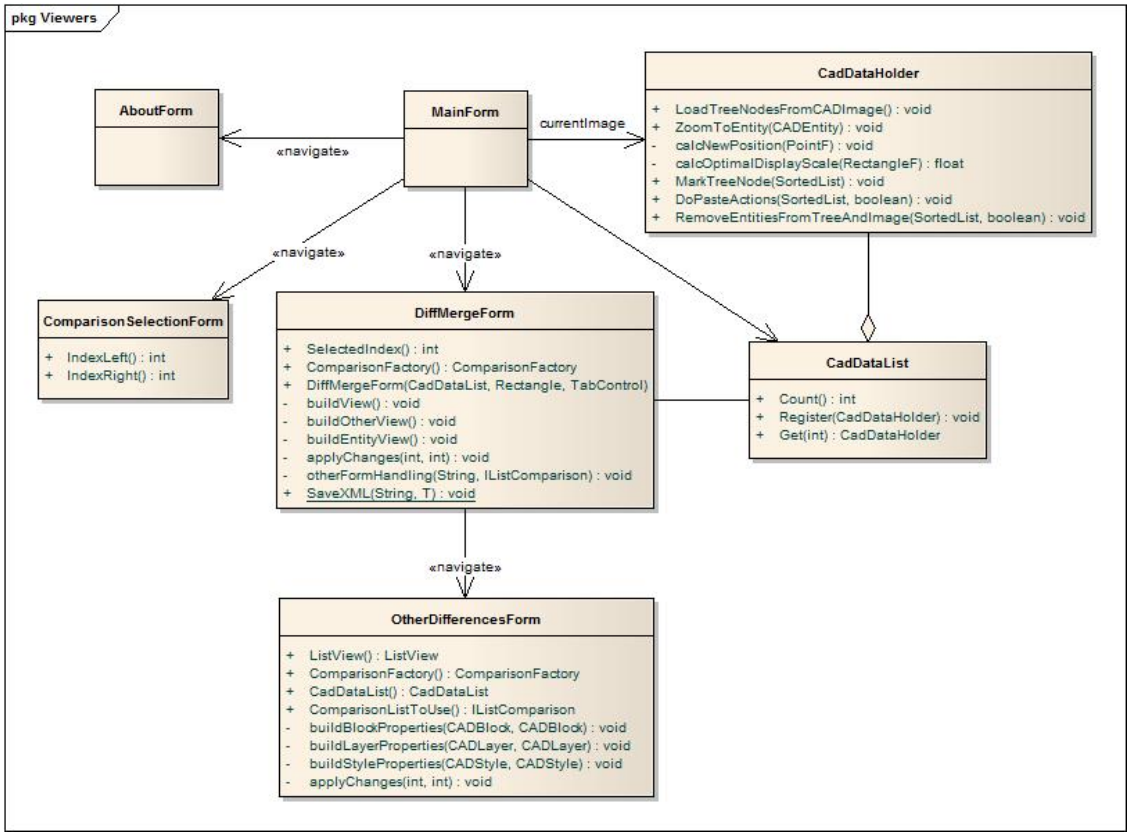

Abbildung 6.4: Package Inhalt Viewers

- den Merge-Modus zur Konfliktauflösung mit drei aktiven Karteireitern.

Im Merge-Modus wird beim Durchführen des Vergleiches die *ComparisonSelectionForm* angezeigt. Dies ist in Abbildung 6.6 dargestellt. Es werden die zu vergleichenden Karteireiter (**Current**, **Other**, **Theirs**) gewählt. Diese werden über die Methoden *IndexLeft()* und *IndexRight()* der *ComparisonFactory* für die Durchführung des Vergleichs zugeführt. Im Diff-Modus sind sinnvolle Default-Werte an dieser Stelle hinterlegt.

Die *DiffMergeForm* zeigt die Vergleichsresultate für Entitäten im Detail, Blöcken, Layers, LTypes (Linetypes) und Styles an (s. auch Abbildung 6.1). Mittels eines Kontext-Menüs können Änderungen in eine der gewählten Zeichnungen übertreagen werden (man kann dies auch tun, um nur die Vergleichsliste zu verkleinern, das Ergebnis muss hinterher nicht gespeichert werden!). Diese Maske erlaubt den Export der Differenzen in eine XML-Datei. Die Details zu Blöcken, Layers, LTypes und Styles werden von der Maske *OtherDifferencesForm* visualisiert.

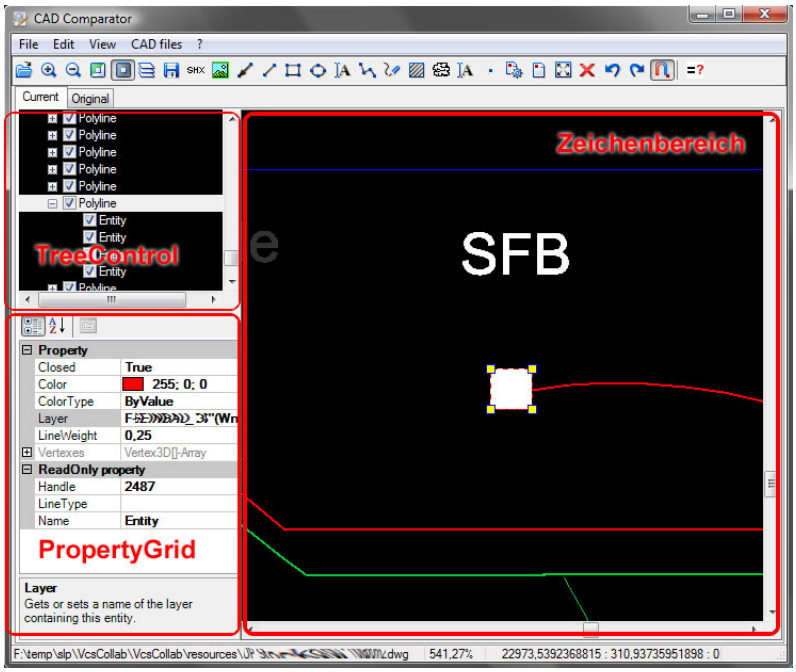

Abbildung 6.5: Bildschirmeinteilung

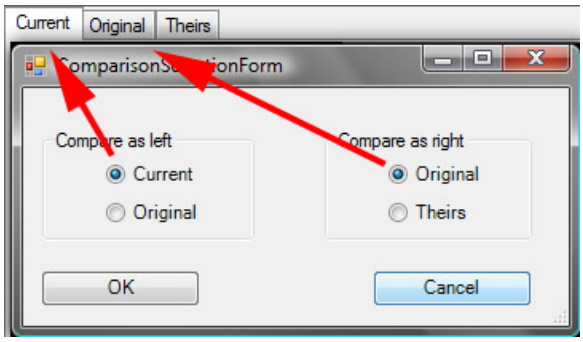

Abbildung 6.6: Auswahl der zu vergleichenden Karteireiter

6.4.3 Schnittstellendesign

Command-Line Schnittstelle

Die Klasse *FileArgUtils* übernimmt das Handling der Command-Line Parameter und exponiert diese mittels statischer Methoden. Es gibt mehrere Möglichkeiten, das Programm aufzurufen.

- Kein Parameter: Das Programm öffnet sich, zeigt zwei Karteireiter an (Current, Other). Der Zeichenbereich ist leer.
- Ein Parameter: Das Programm öffnet sich, zeigt zwei Karteireiter an. Es

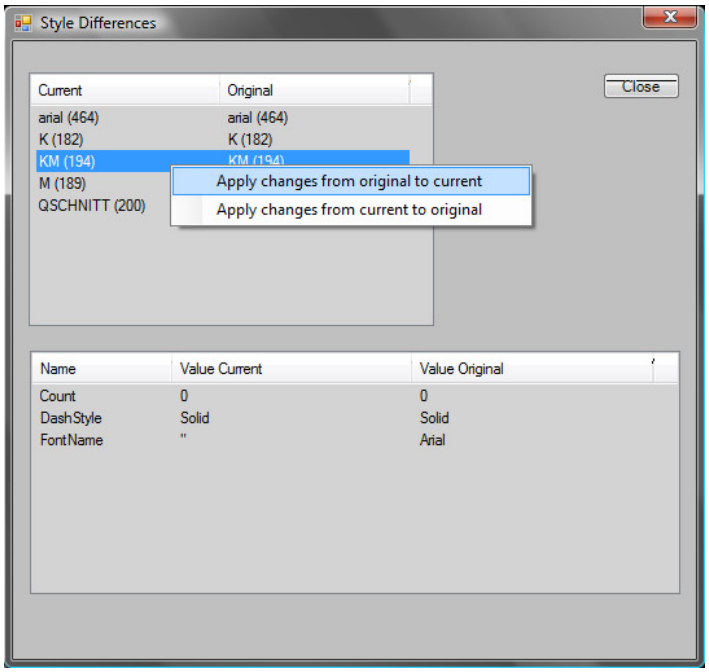

Abbildung 6.7: Anzeige Style Differenzen

wird versucht, die durch den ersten Parameter angegebene Datei in den Karteireiter **Current** zu laden.

- Zwei Parameter (Diff-Mode): Das Programm öffnet sich, zeigt zwei Karteireiter an. Der Karteireiter **Current** wird mit der im **zweiten(!)** Parameter angegeben Datei geladen, der erste Parameter wird für die Beladung des Karteireiters **Other** verwendet. Der Grund dafür ist der Versuch, Fehler bei der Konfiguration von TortoiseSVN zu minimieren. Dort wird für die Registration eines externen Diff-Viewers als erstes der Parameter %base (=Other) und dann der Parameter %mine (=Current) angegeben.

- Drei Parameter: Das Verhalten ist gleich wie bei zwei Parametern. Es wird eine Warnung ausgegeben, dass der dritte Parameter nicht ausgewertet wird.

- Vier Parameter (Merge-Mode, s. auch Arbeitsabläude unter 3.4): Intern wird ein Merge-Mode Flag gesetzt. Es sind drei Karteireiter sichtbar. Der erste Karteireiter (**Current**) wird mit der im dritten, der Karteireiter **Other** mit der im vierten, der Karteireiter **Theirs** mit der im zweiten Parameter genannten Datei beladen. Dies entspricht wiederum der Default-Reihenfolge in TortoiseSVN (s. auch Abbildung 6.8. Der erste Parameter wird für die Vorgabe des Dateinamen beim Speichern des Ergebnisses genutzt (Parameter %merged).

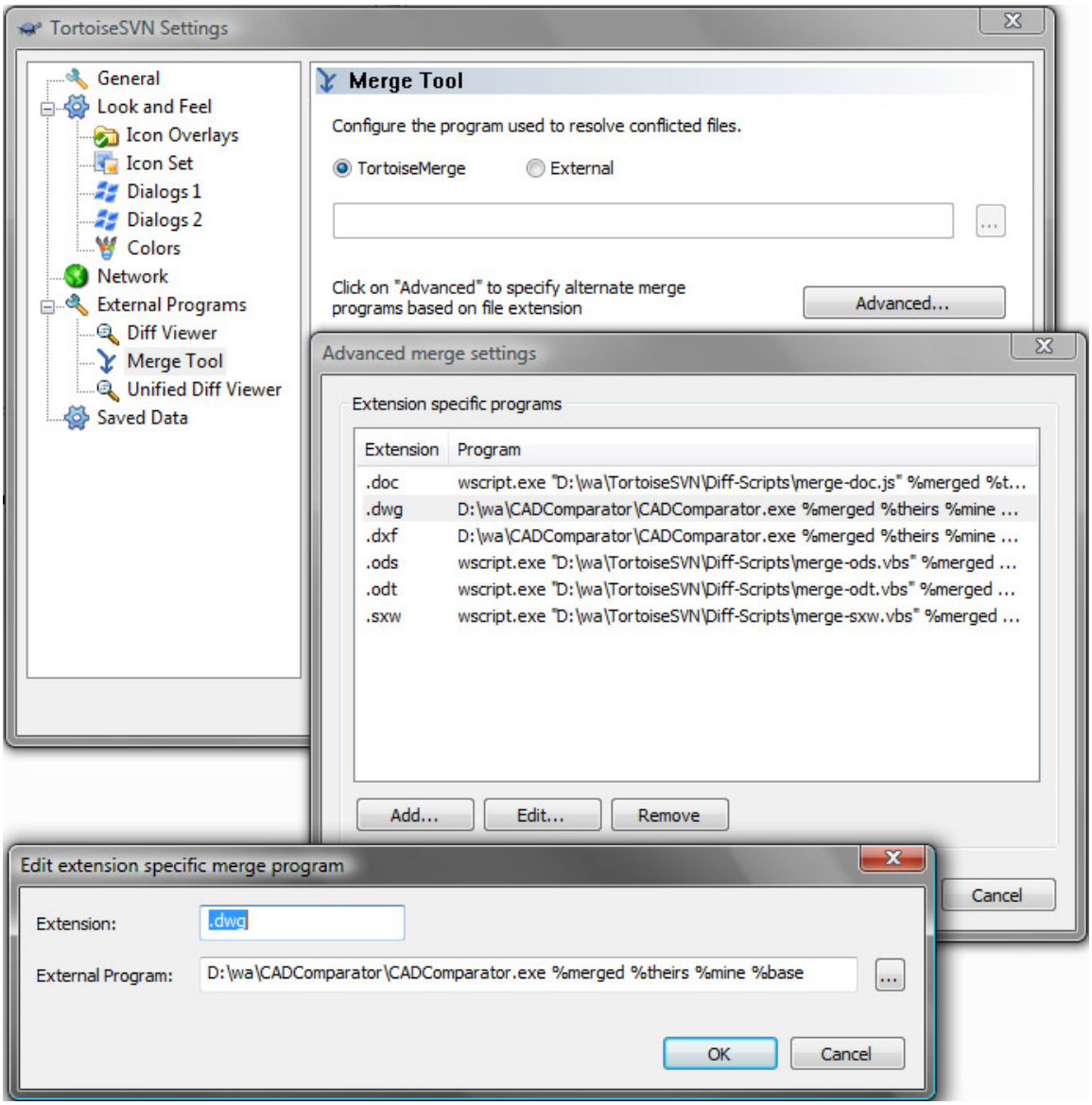

Abbildung 6.8: CADComparator und TortoiseSVN

Die oben gelisteten Feinheiten werden durch die Klasse *FileArgUtils* abstrahiert. Diese bietet Methoden wie *FilenameCurrent* (eigentlich ein Getter) an.

XML Export-Schnittstelle

Gewisse Designdetails der XML Export-Schnittstelle wurden bereits im Abschnitt 6.4.1 genannt. In dieser Sektion wird das XML-Schema *CadComparator.xsd* vorgestellt.

Die Sektionen Blocks, Entities, Layers, LTypes und Styles des Schemas sind strukturell gleich. Sie unterscheiden sich in der untersten Ebene in den Elementen

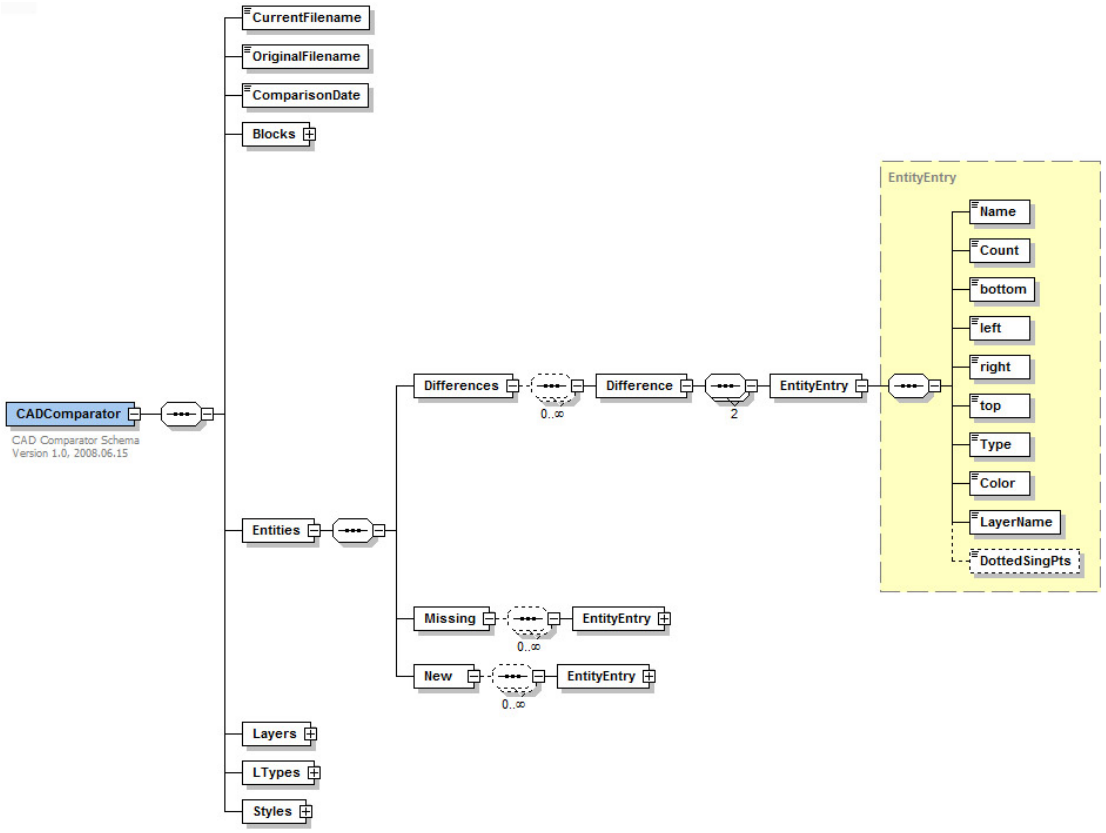

Abbildung 6.9: CADComparator XML Schema

und in der Benennung des hierarchisch darüberliegenden Elementes. So ist in der Abbildung 6.9 auf der rechten Seite der Typ *EntityEntry* (gelber Hintergrund) unter dem Strukturelement *EntityEntry* zu sehen. Analog wäre in der Sektion Blocks der Typ *BlockEntry* unter dem Strukturelement *BlockEntry* anzutreffen.

Das gesamte Schema ist im Anhang A.2 zu finden. Das Schema sollte nach den jeweiligen Bedürfnissen einer weiteren Verarbeitung angepasst werden.

DXF Export-Schnittstelle

Die DXF Export-Schnittstelle wird mit dem Menüpunkt “File->Save as DXF”in der Hauptmaske angestossen.

Kapitel 7

Implementierungsdetails CAD-Plugin

Die folgenden Sektionen beschreiben technische Details der Implementierung.

7.1 Technische Details und Hindernisse

7.1.1 Algorithmus Differenzen

Die Hauptaufgabe der Applikation ist der Vergleich zweier CAD-Pläne. Diese werden mit Hilfe der Bilbiothek CAD Import .Net in den Speicher geladen. Danach liegen die in den Zeichnungen enthaltenen Objekte nach der von der Bibliothek vorgegebenen Klassenhierarchie (getrennt nach Quellen) zur weiteren Verarbeitung vor.

Der Vergleichsalgorithmus sei nun im Detail beschrieben. Dieser operiert auf den typisierten (z.B. Styles, Layers) Objekten im Speicher. Die zwei Quellen werden als A und B angesprochen und Objekte aus Quelle A sind in *listA* und Objekte aus Quelle B sind in **listB** zu finden.

Das Resultat des Algorithmus sind drei Listen, die folgendes enthalten:

- Elemente aus A, die nicht in B enthalten sind.
- Elemente aus B, die nicht in A enthalten sind.
- Elemente, die in A und B enthalten sind, aber nicht als identisch eingestuft wurden.

```
public void DoComparison(SortedList listA, SortedList listB)
{
  Boolean exactMatchFound = processLists(listA, listB,
```

```
    ComparisonType.Identical);

  processLists(listA, listB, ComparisonType.HandleIdentical);
  // assign the rest
  foreach (CADEntity ce in listB.GetValueList())
  {
    this.itemsNotInA.Add(ce);
  }
  foreach (CADEntity ce in listA.GetValueList())
  {
    this.itemsNotInB.Add(ce);
  }
}
```

Der gezeigte Code aus der Klasse *AbstractComparison* zeigt den Ablauf des Vergleichsalgorithmus. Es werden zwei Läufe (Methode *processLists(...)*) durchgeführt. Im ersten Lauf werden nur exakte Vergleiche erlaubt. Damit wird einer versehentlichen Ähnlichkeit entgegengewirkt (eine zuerst auftretende Ähnlichkeit könnte eine in einem späteren Schleifendurchlauf angetroffene Identität überdecken). Die Variable *exactMatchFound* kann zu einer Optimierung des Algorithmus genutzt werden. Falls in zwei Listen zu je beispielsweise 2000 Elementen keine Identitäten aufgefunden werden, ist ein zweiter Durchlauf mit gelockerten Bedingungen (in komplexeren Kombinationen) nicht nötig. Man sollte die Optimierung möglicherweise auf bestimmte Elemente einschränken, die den grössten Zeitgewinn bringen: dies wären die Entitäten - im Gegensatz zu Blöcken, Layers, LTypes und Styles. Die Entitäten machen den Grossteil der Objekte in einer Zeichnung aus.

Ein positiver Vergleich führt zu einer Entfernung des Elementes aus *ListA* und *ListB*. Damit wird das Problem des Algorithmus auch sichtbar: zwei komplett unterschiedliche Listen zu verarbeiten dauert (je nach Grösse) lange, da nie ein positiver Vergleich erzielt wird und damit die ursprünglichen Vergleichslisten *listA* und *listB* nicht kleiner werden. Hier könnte obiger Optimierungsansatz definitiv helfen. Der Rest der Elemente wird zum Schluss *itemsNotInA* bzw. *itemsNotInB* zugeteilt.

Die komplette Methode *processLists(...)* ist im Anhang A.3 zu finden.

7.1.2 Details zu Vergleichskriterien

Die Vergleichskriterien können in primäre und sekundäre Kriterien eingeteilt werden.

- Primäre Kriterien müssen erfüllt sein, damit eine Ähnlichkeit zweier Elemente erkannt wird (*ComparisonType.HandleIdentical*).
- Sind auch die sekundären Kriterien erfüllt, werden die Elemente als identisch bewertet (*ComparisonType.Identical*).

Die Liste der Objekttypen und Vergleichskriterien sind in Tabelle 7.1 gelistet. Eine Kurzbeschreibung der verwendeten Begriffe in der Tabelle folgt:

- Box: vier Koordinaten (bottom, left, right, top).
- Color: Farbe des Elementes.

Objekttyp	primäre Kriterien	sekundäre Kriterien
Block	EntName, EntType	Box, Color, Count
Entity	Box	isMinorMatch(), isAttribMatch()
oder	isTopLeftWidthEqual(), isMinorMatch()	
oder	isTopLeftHeightEqual(), isMinorMatch()	
oder	useTextMatches(), isLeftEqual()	
Layer	EntName, EntType	Box, Color, Frozen, Visibility
LType	EntName, EntType	Box, Color, DashStyle, LineWeight
Style	EntName, EntType	DashStyle, FontName

Tabelle 7.1: Verwendete Vergleichskriterien

- Count: Anzahl gelisteter Sub-Elemente.
- DashStyle: Linien sind durchgezogen, strichliert, ...
- EntName: Name des Elementes; im Falle von Entitäten kann dies der Inhalt eines Textobjektes sein bzw. dem Typ entsprechen.
- EntType: Typ des Elementes (z.B. Block, Layer, ...).
- FontName: Schriftart eines Style.
- Frozen: ein ja/nein Flag eines Layers.
- isAttribMatch: vergleicht Tag und Text eines CADAttrib.
- isLeftEqual: vergleicht das Attribut left (einer Box)
- isMinorMatch: vergleicht einen durch *StripMtext* normalisierten Namen, Count, Color und Layernamen. Im Falle von Linien werden auch die Anzahl der Einzelpunkte (DottedSingPts) einbezogen.
- isTopLeftHeightEqual: vergleicht top, left und height (einer Box).
- isTopLeftWidthEqual: vergleicht top, left und width (einer Box).
- LineWeight: Strichstärke einer Linie.
- Visibility: ein ja/nein Flag eines Layers.

7.1.3 Behandlung von CAD Textelementen

In der Abbildung 7.1 ist ein Textelement zu sehen. Das Element ist vom Typ MText [6]. Das Element erlaubt die Angabe von Inline Formatangaben gemischt mit den Textinformationen.

Ein Beispiel für Inline Formatangaben:

```
b{\Fromans.shx|c1;=}BA{\Fromans.shx|c1;/DB}2B{\Fromans.shx|c1;/}
```

Der reine Text zu obigem Beispiel ist der folgende:

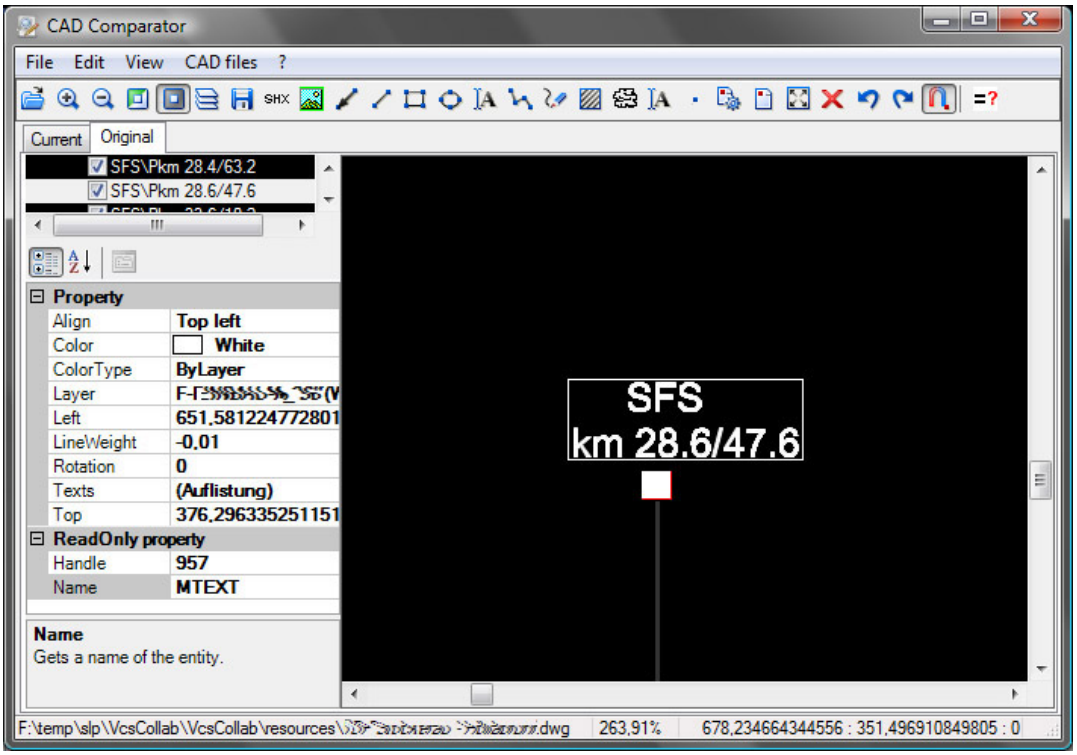

Abbildung 7.1: Illustration Elementtyp MText

```
b=BA/DB2B/
```

Die Extraktion des Textes wird mit folgendem regulären Ausdruck erreicht (C# Notation mit Escaping):

```
string regex = @"\\.*?;(.*?)|\{\\.*?;(.*?)?\}";
```

Die Funktionalität, den Text zu extrahieren, wird in der Applikation an verschiedenen Stellen benötigt und durch die Methode *StripMtext(...)* der Klasse *TextUtils* zur Verfügung gestellt: Für die Anzeige der Texte in der Baumdarstellung, bei der Darstellung der Differenzen und für die Sortierung der Texte (Klasse *EntityNameComparator*). Bei der Prüfung auf Änderungen kommt diese Routine auch zum Einsatz, nachdem bemerkt wurde, dass eine einfache DXF nach DXF Konversion (mit AnyDWG [5]) an dieser Stelle zu Änderungen geführt hatte (aber zu keinen bemerkbaren visuellen Unterschieden). D.h. die Informationen werden “normalisiert ”, damit es nicht zur Anzeige von Änderungen kommt, die der Benutzer unnötigerweise sichten müsste.

7.1.4 Export nach DXF

Die in der Applikation verwendete Bibliothek CAD Import .NET erlaubt auch den Export der angezeigten Darstellung in eine DXF-Datei. Dies funktioniert in der momentanen Version mit diversen Testdateien aber leider nicht. Als ein Workaround könnte derzeit der XML-Export genutzt werden. Dies wäre allerdings mit hohem Aufwand der darauf aufbauenden Applikation verbunden.

Das Problem wurde dem Hersteller gemeldet und es ist zu hoffen, dass dieses in naher Zukunft behoben wird. Die Reaktionszeit des Herstellers der Bibliothek ist leider nicht optimal.

7.2 Kurzanleitung CADComparator

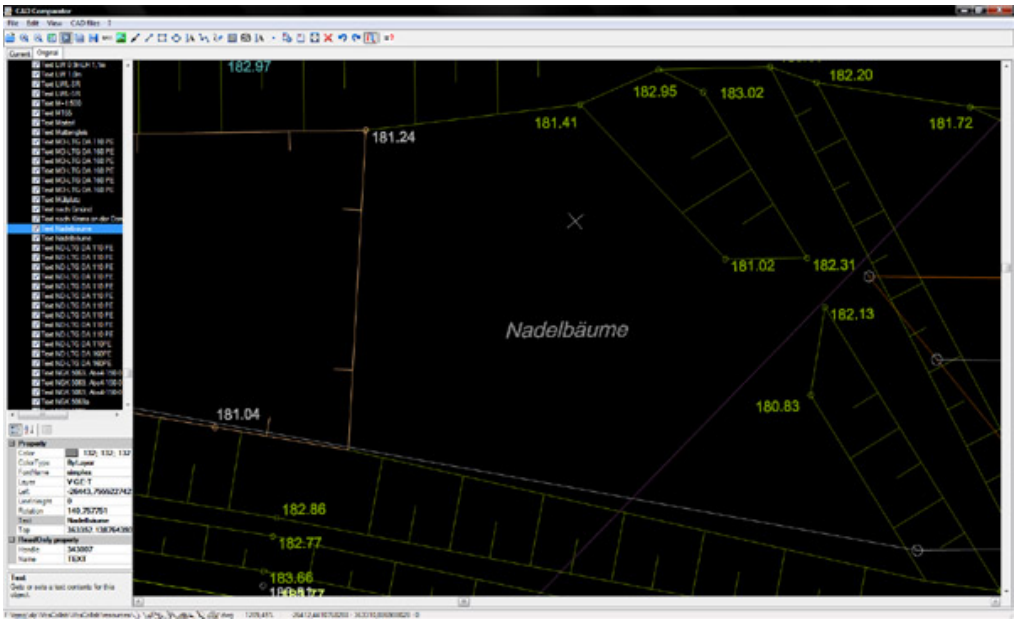

Abbildung 7.2: Zoom-In via Doppelclick im TreeControl

In dieser Sektion wird eine Kurzbeschreibung der wichtigsten Features gegeben.

Wenn CADComparator von TortoiseSVN gestartet wird (wie im Falle des in Abbildung 1.1 illustrierten Ablaufes), öffnet sich die Applikation bereits mit geladenen Zeichnungen. Es wird im folgenden davon ausgegangen, dass zwei Zeichnungen geladen sind (dies entspricht dem Diffenz-Modus) und - zur Illustration gewisser Features - dass Unterschiede zwischen den Zeichnungen bestehen.

Das TreeControl (s. auch Abbildung 6.5) enthält die Elemente der Zeichnung gruppiert nach Blöcken, Entitäten, Layers, LTypes und Styles. Abgesehen von dieser Präsentations-Einteilung im Plugin ist die Struktur unter den fünf Top-Level Typen flach und wird alphabetisch sortiert angezeigt. D.h. im TreeControl werden nicht Layer- oder Block-Strukturen nachgebildet. Entitäten und Layers (diese sind weiss dargestellt; die anderen Elemente werden türkis angezeigt) erlauben eine detallierte Anzeige und Editiermöglichkeit im PropertyGrid. Eine *Selektion im TreeControl* bewirkt das *Zentrieren* des entsprechenden Elementes im Zeichenbereich. Ein *Doppelclick* führt zu einem *Zoom-In* auf das Element (Eine Beispieldarstellung ist in Abbildung 7.2 zu sehen).

Umgekehrt führt eine Selektion eines Elementes im Zeichenbereich zu einer Markierung im TreeControl. Auf diese Weise bleibt die Übersicht für den Benutzer gewahrt.

Die Anzeige der Differenzen kann entweder über das Menü (Tastaturkürzel CTRL+O) oder über das Symbol ganz rechts in der Symbolleiste aufgerufen werden.

Es wird jeweils zum gewählten Element gezoomt bzw. das gewählte Element zentriert (dies ist über eine Checkbox einstellbar, s. auch Abbildung 6.1). Falls ein Element nur in einer Zeichnung vorkommt, wird bei Selektion automatisch der entsprechende Karteireiter aktiv. Falls das Element in beiden Zeichnungen (aber geändert) vorkommt, kann mit *Doppelclick* auf die Zeile der Karteireiter gewechselt werden.

Ein Kontextmenü erlaubt in der Differenzanzeige das Übertragen von Änderungen. Um die Liste zu verkleinern (bzw. um gewisse Änderungen als gesehen auszuschliessen), können Änderungen auch z.B. in das Original zurückübertragen werden. Es muss nicht gespeichert werden, die Änderungsliste wird aber entsprechend übersichtlicher.

Die angezeigten Differenzen können als Schema constrained XML exportiert werden.

7.3 Limits und Erweiterungsmöglichkeiten

Die aktuellen Limits der Implementierung sind folgende:

- Speichern als DXF funktioniert nicht zuverlässig und ist ein Problem in der verwendeten Bibltiothek.
- Folgende Objekttypen werden momentan nicht unterstützt (weder Vergleich noch Abgleich): Viewports (Ansichten; in der DXF Spezifikation *VPorts* genannt), Layouts (traditionell "Papierbereich") und XRefs (external reference files, also Verweise aller Art). Diese Entscheidung wurde aufgrund der involvierten Komplexität (dies betrifft vor allem XRefs) getroffen.
- Es ist (teilweise aufgrund des vorigen Punktes) nicht möglich, einen Abgleich zweier komplett verschiedener Zeichnungen durchzuführen.

Als Erweiterungsmöglichkeiten sind vorstellbar:

- Visualisierung von Block-, LType- und Style-Attributen im PropertyGrid der Applikation (diese Elemente sind momentan im TreeControl nur mit Namen gelistet).
- Eine Commandline-Option zum automatischen Anstossen des XML-Exports.
- Die Visualisierung von Rasterbildern ist bereits möglich. Die Darstellung von Unterschieden zwischen Rasterbildern wäre bei Interesse eine weitere Option, die relativ leicht eingebaut werden könnte.
- Eine Anbindung an ein DMS (z.B. Alfresco) oder andere VCS mit dem Ziel, verschiedene Versionen eines Dokumentes zu vergleichen.

7.4 CADComparator und Open Source

Es ist geplant, das Werkzeug in die Open Source Community (z.B: auf *SourceForge* oder *CodePlex*) einzubringen und weiterhin zu pflegen. Die Designs und Quellcode des Plugins werden gerne zur Verfügung gestellt.

Der Hersteller der Bibliothek erlaubt das gemeinsame Packaging mit der Dmeo-Version der CAD Import .Net Bibliothek (also ohne eincodierte Registrierung). Ohne Registrierung der Bibliothek ist kein Batch-Betrieb möglich, da ein Bestätigungsfenster nach Öffnen einer Zeichnung angezeigt wird, welches den Demo-Zustand der Bibliothek festhält.

7.5 Vergleich mit existierenden Lösungen

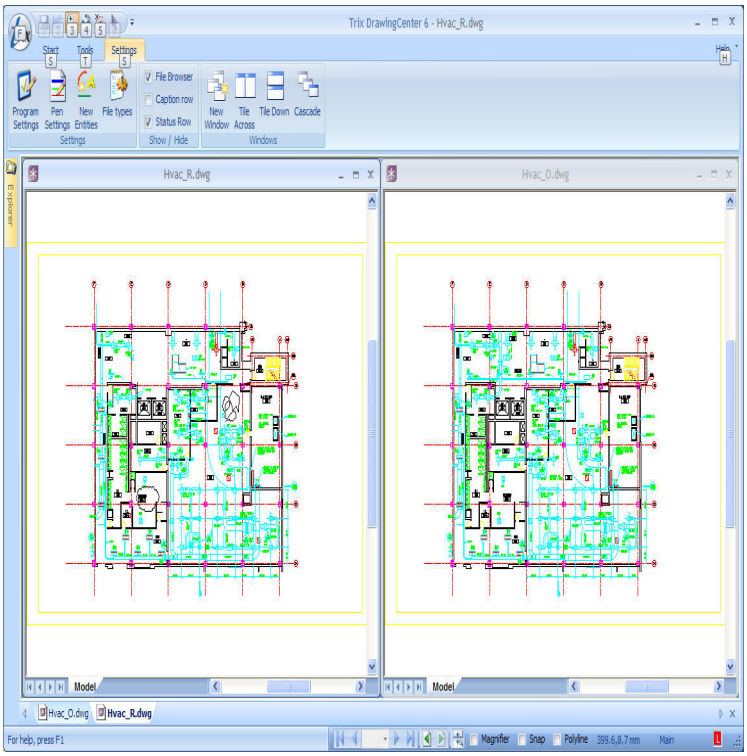

Abbildung 7.3: Trix DrawingCenter / TracTrix: Vergleich von Zeichnungen

Es existieren natürlich kommerzielle Lösungen für den in dieser Arbeit beschriebenen praktischen Ansatz. Die Abbildung 7.3 zeigt das *Trix DrawingCenter* von Trix Systems Inc. [23]. Das Produkt ist ohne eine AutoCAD Installation lauffähig und bietet viele Funktionen. Das Ergebnis eines Vergleiches von Zeichnungen wird in Form Bildes visualisiert, wobei die einzelnen Veränderungen unterschiedlich eingefärbt sind (Abbildung 7.4).

Ein weiteres Produkt, das Unterschiede in CAD Zeichnungen visualisieren kann, ist *DWG Compare* der Firma Overcad [37]. Die Nutzung von DWG Compare setzt allerdings eine Installation von AutoCAD voraus. Das Produkt wird als AutoCAD Runtime Extension (ARX) ausgeliefert.

Der in dieser Arbeit vorgestellte Ansatz verfolgt einfache Zielsetzungen:

- Das Ergebnis soll frei verfügbar sein, um für Änderungen und Erweiterungen offen zu stehen (s. auch 7.3).
- CADComparator soll ein einfaches Werkzeug sein: Die Installation umfasst 4 Dateien (mitsamt der XML-Schema Definition).
- Das Plugin bietet Abgleichsmöglichkeiten.
- Die Unterschiede sind als XML exportierbar.

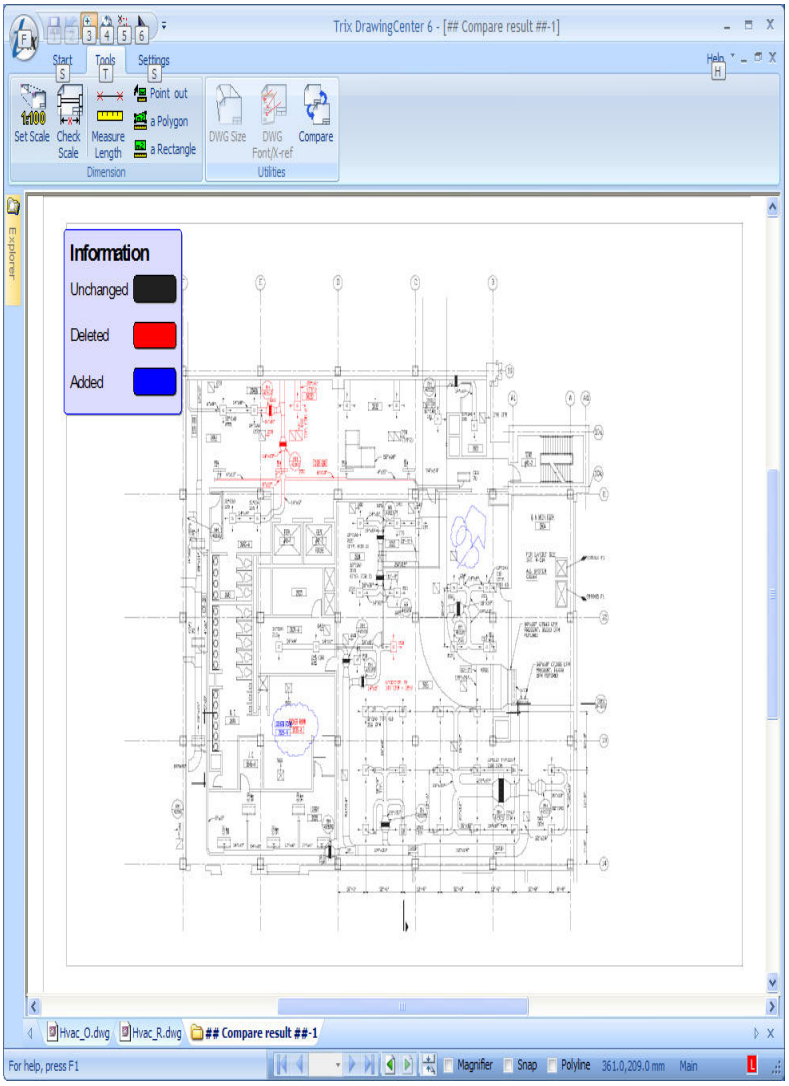

Abbildung 7.4: Trix DrawingCenter / TracTrix: Unterschiede

Kapitel 8

Conclusio

Dateibasiertes kooperatives Arbeiten ist mit organisatorischer Unterstützung am besten umsetzbar. Damit ist gemeint, dass als Ergänzung zu rein technischen Lösungen ein geregelter Arbeitsablauf bzw. Prozeß verteiltes Arbeiten ermöglicht. Dieser Ablauf kann, wie auch ausgeführt worden ist, durch VCS und DMS erheblich verbessert werden. Exklusive Dateisperren durch den gerade aktiven Bearbeiter weisen andere Versuche, die gleiche Datei zu bearbeiten, ab. Dateisperren sind allerdings kooperativem Arbeiten nicht zuträglich. Workflows, die im Rahmen von DMS definierbar sind, strukturieren die gemeinsame Bearbeitung von Dateien, indem beispielsweise Stationen festgelegt werden, die ein Dokument durchlaufen muss, um den endgültigen Bearbeitungszustand zu erreichen. Ein Einsatz von DMS wird im Falle von komplexeren Abläufen bei der Bearbeitung von Dokumenten einem Einsatz von VCS vorzuziehen sein.

Die Abläufe in den Bereichen Software-Engineering, Bearbeitung von Dokumenten allgemein und CAD-Plänen im speziellen zeigten die Eignung von VCS bzw. DMS für den jeweiligen Bereich auf (Kapitel 3.4). CAD-Pläne vereinigen aufgrund ihrer Komplexität die Anforderungen, die an ein VCS gestellt werden (=gleichzeitiges Bearbeiten), und mitunter auch die Anforderungen, die von DMS gut unterstützt werden (=Workflows).

Der technische Aspekt wurde praktisch beleuchtet: Eine effiziente Konfliktresolution im Falle von Dateiformaten wie DWG und DXF, die im Bereich CAD eingesetzt werden, ist mit einfachen Mitteln nicht möglich. Das Format lässt es nicht zu, anhand einer einfachen textuellen Darstellung der Unterschiede eine Entscheidung zu treffen, welche Version zum gewünschten Endergebnis der Bearbeitungen erhoben werden soll. Für diesen Zweck wurde im Rahmen dieser Arbeit ein Plugin erstellt, um die verteilte Bearbeitung von CAD-Dateien zu unterstützen. Dies wurde durch die Integration des Plugins mit einem Versionskontrollsystem erreicht und durch die Visualisierung von Unterschieden zwischen zwei Versionen einer Zeichnung. Dies unterstützt den Benutzer bei einer Bewertung von Änderungen.

Sowohl ein rein technischer Ansatz als auch ein rein organisatorischer Ansatz sind nicht ausreichend für eine effiziente kooperative Bearbeitung von Dateien in einer komplexeren

Arbeitsumgebungt. Einfaches Textformate, wie sie im Umfeld der Programmierung in Form von Quellcode existieren, sind noch mit Mitteln handhabbar, die VCS out-of-the-box anbietet. Allerdings wird auch dort schon im Falle von verteilten Änderungen an gleichen Dateien bald eine Komplexität erreicht, die nur mit organisatorischen Mitteln reduziert werden kann.

Organisatorische Vorgaben kombiniert mit den technischen Möglichkeiten, die VCS bzw. DMS und die Visualisierung in einer natürlichen Applikatonsumgebung bieten, helfen die Zusammenarbeit zu verbessern.

Ob ein VCS oder ein DMS auf technischer Seite die Basis für kooperatives Arbeiten bilden soll, ist im konkreten Falle einer Analyse zu unterziehen.

Literaturverzeichnis

[1] Alfresco. Collaborative content production. http://wiki.alfresco.com/wiki/Collaborative_Content_Production, Februar 2008. Zugriff am 17.03.2008. [zitiert auf S. 26]

[2] Alfresco. Transparent layers. http://wiki.alfresco.com/wiki/Transparent_Layers, Februar 2008. Zugriff am 17.03.2008. [zitiert auf S. 26]

[3] Inc. Alfresco Software. Alfresco. http://www.alfresco.com/, August 2007. Zugriff am 02.12.2007. [zitiert auf S. 22, 25]

[4] EricKow (alias). darcs: distributed, interactive smart. http://wiki.darcs.net/DarcsWiki, November 2007. Zugriff am 21.11.2007. [zitiert auf S. 7]

[5] Inc AnyDWG Software. Any dwg dxf converter. http://anydwg.com/dwg-dxf/, Januar 2008. Zugriff am 08.06.2008. [zitiert auf S. 44, 58]

[6] Autodesk. Mtext [dxf reference]. http://www.autodesk.com/techpubs/autocad/acad2000/dxf/mtext_dxf_06.htm, 2000. Zugriff am 14.06.2008. [zitiert auf S. 57]

[7] Petr Baudis. Git - svn crash course. http://git.or.cz/course/svn.html. Zugriff am 24.02.2008. [zitiert auf S. 35, 85]

[8] Petr Baudis. Git - fast version control system. http://git.or.cz/, 2008. Zugriff am 24.02.2008. [zitiert auf S. 34]

[9] C. Michael Pilato Ben Collins-Sussman, Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly, Juni 2004. [zitiert auf S. 17]

[10] C. Michael Pilato Ben Collins-Sussman, Brian W. Fitzpatrick. Version control with subversion (for subversion 1.4). http://svnbook.red-bean.com/en/1.4/index.html, 2007. Zugriff am 02.12.2007. [zitiert auf S. 20, 21, 22]

[11] Daniel Berlin and Garrett Rooney. *Practical Subversion*. Apress, Mai 2006. [zitiert auf S. 17]

[12] Chris Birmele. Branch and merge primer. http://msdn.microsoft.com/en-us/library/aa730834(VS.80).aspx, August 2006. Zugriff am 26.08.2008. [zitiert auf S. 24]

[13] CollabNet. Tortoisesvn. http://tortoisesvn.tigris.org/, July 2008. Zugriff am 27.08.2008. [zitiert auf S. 42]

[14] CollabNet Corporation. sharpsvn. http://sharpsvn.open.collab.net/, 2008. Zugriff am 10.06.2008. [zitiert auf S. 32]

[15] DarcsWiki. Preparation branches. http://wiki.darcs.net/DarcsWiki/PreparationBranches, Dezember 2007. Zugriff am 26.08.2008. [zitiert auf S. 33]

[16] Andrew Hunt David Thomas. *Pragmatic Version Control Using CVS*. The Pragmatic Programmers LLC, September 2003. [zitiert auf S. 17]

[17] Markus Nix et al. *Web Content Management. CMS verstehen und auswählen*. entwickler.press, 2005. [zitiert auf S. 18]

[18] Per Cederqvist et al. Version management with cvs - the cvs manual. http://www.network-theory.co.uk/docs/cvsmanual/Binaryhowto.html, 2002. Zugriff am 26.08.2008. [zitiert auf S. 32]

[19] The Apache Software Foundation. Converting from cvs to subversion at the asf. http://www.apache.org/dev/cvs2svn.html, 2007. Zugriff am 29.02.2008. [zitiert auf S. 31]

[20] A.-W. Scheer G. Keller, M. Nüttgens. Semantische prozeßmodellierung auf der grundlage ereignisgesteuerter prozeßketten (epk). http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf, Januar 1992. Zugriff am 02.12.2007. [zitiert auf S. 12]

[21] Edward Holness. Ooosvn v0.38. http://sourceforge.net/projects/ooosvn/, 2007. Zugriff am 03.12.2007. [zitiert auf S. 26]

[22] Edward Holness. Odf-svn v1.0a1. http://sourceforge.net/projects/odfsvn, 2008. Zugriff am 26.08.2008. [zitiert auf S. 28]

[23] Trix Systems Inc. Compare two versions of a drawing. http://www.trixsystems.com/compare-drawings.html, 2008. Zugriff am 27.06.2008. [zitiert auf S. 61]

[24] Inc. Interface 21. Acegi security system for spring. http://acegisecurity.org/, 2007. Zugriff am 28.01.2008. [zitiert auf S. 21]

[25] The JBoss jBPM Team. Jboss business process management. http://labs.jboss.com/jbossjbpm/, 2007. Zugriff am 02.12.2007. [zitiert auf S. 21, 22]

[26] Gaurav Kathuria. *Web Content Management with Documentum*. PACKT Publiching, 2006. [zitiert auf S. 20, 21]

[27] March Hare Pty Ltd. Mergepoint. http://www.cvsnt.org/wiki/MergePoint, November 2003. Zugriff am 29.02.2008. [zitiert auf S. 31]

[28] March Hare Pty Ltd. Comparing cvs and cvsnt. http://march-hare.com/cvspro/compare.htm, März 2005. Zugriff am 29.02.2008. [zitiert auf S. 30, 31, 85]

[29] Soft Gold Ltd. Cad import .net. http://www.cadsofttools.com/en/products/cad_import_.net.html, Januar 2008. Zugriff am 08.06.2008. [zitiert auf S. 42]

[30] Mike Mason. *Pragmatic Version Control Using Subversion.* The Pragmatic Programmers LLC, 2nd edition, Mai 2006. [zitiert auf S. 17]

[31] Josh McDonald. xdelta. http://xdelta.org/, Aprilr 2009. Zugriff am 10.06.2008. [zitiert auf S. 32]

[32] Sun Microsystems. Openoffice.org. http://www.openoffice.org/, 2007. Zugriff am 03.12.2007. [zitiert auf S. 26]

[33] Tom Moertel. How i stopped missing darcs and started loving git. http://blog.moertel.com/articles/2007/12/10/how-i-stopped-missing-darcs-and-started-loving-git, December 2007. Zugriff am 26.08.2008. [zitiert auf S. 37]

[34] Jakub Narebski. Projects that use git ... http://git.or.cz/gitwiki/GitProjects, Februar 2008. Zugriff am 02.03.2008. [zitiert auf S. 34]

[35] David Neary. Subversion - a better cvs. http://www.linux.ie/articles/subversion/, September 2005. Zugriff am 29.02.2008. [zitiert auf S. 31]

[36] Oracle. Oracle berkeley db product family. http://www.oracle.com/technology/products/berkeley-db/index.html, 2008. Zugriff am 26.01.2008. [zitiert auf S. 19]

[37] OverCAD. Dwg compare. http://www.overcad.com/dwg-compare.php, 2008. Zugriff am 27.06.2008. [zitiert auf S. 61]

[38] Mark Phippard. Subversion 1.45. http://subversion.tigris.org/, August 2007. Zugriff am 02.12.2007. [zitiert auf S. 21, 31, 32]

[39] Kay Ramme. Universal network objects (openoffice). http://udk.openoffice.org/, 2007. Zugriff am 02.12.2007. [zitiert auf S. 28]

[40] Inc. Red Hat. Cygwin. http://www.cygwin.com/, 2007. Zugriff am 03.12.2007. [zitiert auf S. 26]

[41] David Roundy. Getting started with darcs. http://wiki.darcs.net/DarcsWiki/GettingStarted, November 2007. Zugriff am 29.02.2008. [zitiert auf S. 32]

[42] David Roundy. (darcs) introduction. http://www.darcs.net/manual/node9.html, August 2008. Zugriff am 27.08.2008. [zitiert auf S. 30, 38]

[43] Munwar Shariff. *Alfresco: Enterprise Content Management Implementation.* PACKT Publishing, 2006. [zitiert auf S. 22]

[44] TMate Software. Subversioning for java. http://svnkit.com/, 2008. Zugriff am 29.02.2008. [zitiert auf S. 32]

[45] Bryan O Sullivan. (mercurial) introduction. http://hgbook.red-bean.com/, December 2007. Zugriff am 02.03.2008. [zitiert auf S. 34, 40]

[46] The CVS Team. Concurrent versions system - neuigkeiten. http://savannah.nongnu.org/news/?group=cvs, 2006. Zugriff am 29.02.2008. [zitiert auf S. 30]

[47] Wikipedia. Autocad. http://de.wikipedia.org/wiki/AutoCAD, November 2007. Zugriff am 21.11.2007. [zitiert auf S. 7]

[48] Wikipedia. Dokumentenmanagement. http://de.wikipedia.org/wiki/Dokumentenmanagement, Dezember 2007. Zugriff am 27.01.2008. [zitiert auf S. 18]

[49] Wikipedia. Office open xml. http://de.wikipedia.org/wiki/Office_Open_XML, Oktober 2007. Zugriff am 21.11.2007. [zitiert auf S. 8]

[50] Wikipedia. Patch (unix). http://de.wikipedia.org/wiki/Patch_(Unix), 2007. Zugriff am 28.01.2008. [zitiert auf S. 22]

[51] Wikipedia. Comparison of revision control software. http://en.wikipedia.org/wiki/Concurrent_Versions_System, 2008. Zugriff am 29.02.2008. [zitiert auf S. 30]

[52] Wikipedia. Comparison of revision control software. http://en.wikipedia.org/wiki/Comparison_of_revision_control_software, 2008. Zugriff am 26.02.2008. [zitiert auf S. 31, 32, 34]

[53] Wikipedia. Darcs. http://en.wikipedia.org/wiki/Darcs, August 2008. Zugriff am 26.08.2008. [zitiert auf S. 33, 40]

[54] Wikipedia. Oracle forms. http://en.wikipedia.org/wiki/Oracle_Forms, August 2008. Zugriff am 26.08.2008. [zitiert auf S. 23]

[55] Wikipedia. Version (software). http://de.wikipedia.org/wiki/Version_(Software), 2008. Zugriff am 26.08.2008. [zitiert auf S. 13]

[56] Wikipedia. Versionsverwaltung. http://de.wikipedia.org/wiki/Versionsverwaltung, 2008. Zugriff am 22.01.2008. [zitiert auf S. 17]

[57] Eric Wong. git-svn(1) manual page. http://www.kernel.org/pub/software/scm/git/docs/git-svn.html, Februar 2008. Zugriff am 02.03.2008. [zitiert auf S. 34]

Anhang A

Anhang Technische Details

A.1 OooSVN

Es wurden bei der Untersuchung des OooSVN Plugins v0.38 unter Windows XP mit Service Pack 2, Cygwin 1.5.24-2, SVN 1.4.5 und OpenOffice 2.3 folgende Probleme gefunden, welche die reibungslose Verwendung etwas einschränken können. Diese Probleme treten auch unter Linux auf, ausser es wird explizit darauf hingewiesen.

- Leerzeichen in Datei- bzw. Verzeichnisnamen werden nicht sauber unterstützt. Es ist viel an Funktionalität in externen Shell-Scripts vorzufinden. Die oft an solche Skripts übergebenen Dateinamen mit Pfadangabe sind aber beispielsweise nicht durch Anführungszeichen umgeben. Dies führt zu Fehlfunktionen. Um dies zu umgehen, sind (vor allem unter Windows) die Funktionen *getRepo()* und *getWorkingDir()* in der Datei *Module1.xba* zu ändern. Von einer Verwendung von Leerzeichen für OpenOffice-Dokumentnamen wird bei der aktuellen Version des Plugins auch stark abgeraten.

  Eine Möglichkeit für Windows wäre:

  ```
  Function getRepo()
    getRepo = "file:///c:/temp/.ooosvn"
  end Function

  Function getWorkingDir()
    getWorkingDir = "c:/temp/.ooosvn"
  end Function
  ```

  Es ist möglich, einen nicht lokalen URL für das Repository zu verwenden (z.B. *https://ooosvn.svn.sourceforge.net/svnroot/ooosvn*). Man beachte, dass Schreib-Berechtigungen benötigt werden.

- (Windows only) Unter Windows (mit Cygwin) ist zu beachten, dass der Pfad zu den Cygwin Dateien vor dem Windows Systempfad aufscheint. Konkret betrifft

es den Befehl *find* in der Datei *listdocuments.sh*: Die Version, die in Windows standardmässig verfügbar ist, funktioniert in diesem Kontext nicht.

- (Windows only) Die OpenOffice Basic Routinen des Plugins enthalten mehrfach die Anweisung *mid(svnUrl, 8)*. Dieses Kommando schneidet URL-Prefix Informationen (nämlich die Protokollangabe *file://*) ab. In dieser Form funktioniert das Plugin aber unter Windows nicht. Unter Windows wird die folgende Anweisung benötigt: *mid(svnUrl, **9**)*. Es muss ein Zeichen mehr abgeschnitten werden, falls man diese Art der Programmierung weiterverfolgt. In diesem Zusammenhang ist eine Reduktion von Codewiederholung zu empfehlen (ca. 12 Stellen mussten geändert werden), damit eine einfache Änderung möglich wird.

- Das Herunterladen des Userguide aus dem SVN-Repository via HTTPS funktioniert nicht, da das Zertifikat nicht automatisch akzeptiert werden kann. Dies müsste manuell beispielsweise durch

  ```
  svn checkout
     https://ooosvn.svn.sourceforge.net/svnroot/ooosvn/userguide
  ```

  getan werden. Hierbei sollte das Zertifikat permanent akzeptiert werden, falls man auf den (sehr spärlichen) Userguide nicht verzichten mag. Es sollte in diesem Zusammenhang auch das Skript *userguide.sh* adaptiert werden, da genanntes Problem die Funktion beeinträchtigt. Die korrigierte Version ist im folgenden wiedergegeben:

  ```
  #!/bin/bash
  cd $1/
  rm -rf $1/userguide
  svn checkout $2

  # START added lines
  USERGUIDEFILES=‘ls userguide 2>/dev/null‘
  if [ -z "$USERGUIDEFILES" ]
  then
       # no  files here.. bail out !
       exit 0
  fi
  # END added lines

  cd $1/userguide
  zip -rDX ../userguide.odt * -x *.svn*
  ```

- Die Skripts *import.sh* und *commit.sh* enthalten jeweils eine Anweisung, die den Aufruf *unzip* enthält. Hier sollte die Option *-q* verwendet werden, da ansonsten unerwünschte Inhalte in Dateien gelangen können.

A.2 XML-Schema CadComparator

CadComparator.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://at.ac.tuwien.geoinfo/cadcomparator/1.1/"
  targetNamespace="http://at.ac.tuwien.geoinfo/cadcomparator/1.1/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
      <xs:documentation>Version 1.1, 2008.06.15</xs:documentation>
  </xs:annotation>
  <xs:complexType name="BlockEntry">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Count" type="xs:int"/>
      <xs:element name="bottom" type="xs:float"/>
      <xs:element name="left" type="xs:float"/>
      <xs:element name="right" type="xs:float"/>
      <xs:element name="top" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="EntityEntry">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Count" type="xs:int"/>
      <xs:element name="bottom" type="xs:float"/>
      <xs:element name="left" type="xs:float"/>
      <xs:element name="right" type="xs:float"/>
      <xs:element name="top" type="xs:float"/>
      <xs:element name="Type" type="xs:string"/>
      <xs:element name="Color" type="xs:string"/>
      <xs:element name="LayerName" type="xs:string"/>
      <xs:element name="DottedSingPts" type="xs:int" minOccurs="0"/>
      <xs:element name="Attrs" minOccurs="0">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Attr">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="name" type="xs:string"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="LayerEntry">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Color" type="xs:string"/>
    <xs:element name="Frozen" type="xs:boolean"/>
    <xs:element name="Visibility" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="LTypeEntry">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Color" type="xs:string"/>
    <xs:element name="DashStyle" type="xs:string"/>
    <xs:element name="LineWeight" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="StyleEntry">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="DashStyle" type="xs:string"/>
    <xs:element name="FontName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="CADComparator">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CurrentFilename" type="xs:string"/>
      <xs:element name="OriginalFilename" type="xs:string"/>
      <xs:element name="ComparisonDate" type="xs:dateTime"/>
      <xs:element name="Blocks">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Differences">
              <xs:complexType>
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="Difference">
                    <xs:complexType>
                      <xs:sequence minOccurs="2" maxOccurs="2">
                        <xs:element name="BlockEntry" type="cc:BlockEntry"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
```

```
      </xs:element>
      <xs:element name="Missing">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="BlockEntry" type="cc:BlockEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="New">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="BlockEntry" type="cc:BlockEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Entities">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Differences">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Difference">
              <xs:complexType>
                <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:element name="EntityEntry" type="cc:EntityEntry"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Missing">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="EntityEntry" type="cc:EntityEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="New">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="EntityEntry" type="cc:EntityEntry"/>
          </xs:sequence>
        </xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Layers">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Differences">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Difference">
              <xs:complexType>
                <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:element name="LayerEntry" type="cc:LayerEntry"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Missing">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="LayerEntry" type="cc:LayerEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="New">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="LayerEntry" type="cc:LayerEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="LTypes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Differences">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Difference">
              <xs:complexType>
                <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:element name="LTypeEntry" type="cc:LTypeEntry"/>
```

```
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Missing">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="LTypeEntry" type="cc:LTypeEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="New">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="LTypeEntry" type="cc:LTypeEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Styles">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Differences">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Difference">
              <xs:complexType>
                <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:element name="StyleEntry" type="cc:StyleEntry"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Missing">
        <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="StyleEntry" type="cc:StyleEntry"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="New">
```

```
              <xs:complexType>
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                  <xs:element name="StyleEntry" type="cc:StyleEntry"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Eine Beispiel-Instanz:

```
<?xml version="1.0"?>
<CADComparator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:schemaLocation="http://at.ac.tuwien.geoinfo/cadcomparator/1.1/
  CadComparator.xsd"
  xmlns="http://at.ac.tuwien.geoinfo/cadcomparator/1.1/">
  <CurrentFilename>Test.dwg</CurrentFilename>
  <OriginalFilename>Test2.dwg</OriginalFilename>
  <ComparisonDate>2008-06-13T00:26:19.3463095+02:00</ComparisonDate>
  <Blocks>
    <Differences />
    <Missing />
    <New />
  </Blocks>
  <Entities>
    <Differences>
      <Difference>
        <EntityEntry>
          <Name><![CDATA[Line]]></Name>
          <Count>0</Count>
          <bottom>290.32288380155</bottom>
          <left>887.142406140205</left>
          <right>1520.90382168869</right>
          <top>290.32288380155</top>
          <Type>Line</Type>
          <Color>2fffffff</Color>
          <LayerName><![CDATA[LayerA]]></LayerName>
          <DottedSingPts>2</DottedSingPts>
        </EntityEntry>
        <EntityEntry>
          <Name><![CDATA[Line]]></Name>
          <Count>0</Count>
```

```
        <bottom>290.32288380155</bottom>
        <left>887.142406140205</left>
        <right>1520.90382168869</right>
        <top>290.32288380155</top>
        <Type>Line</Type>
        <Color>2fffffff</Color>
        <LayerName><![CDATA[LayerA]]></LayerName>
        <DottedSingPts>250</DottedSingPts>
      </EntityEntry>
    </Difference>
  </Differences>
  <Missing>
    <EntityEntry>
      <Name><![CDATA[Line]]></Name>
      <Count>0</Count>
      <bottom>316.001687374734</bottom>
      <left>-2156.69281324256</left>
      <right>-2058.66027317281</right>
      <top>322.101066969183</top>
      <Type>Line</Type>
      <Color>2fffffff</Color>
      <LayerName><![CDATA[EINBAU]]></LayerName>
      <DottedSingPts>2</DottedSingPts>
    </EntityEntry>
  </Missing>
  <New>
    <EntityEntry>
      <Name><![CDATA[Line]]></Name>
      <Count>0</Count>
      <bottom>273.708163031888</bottom>
      <left>-2092.67489648025</left>
      <right>-1994.64235641051</right>
      <top>279.807542626337</top>
      <Type>Line</Type>
      <Color>2fffffff</Color>
      <LayerName><![CDATA[EINBAU]]></LayerName>
      <DottedSingPts>2</DottedSingPts>
    </EntityEntry>
  </New>
</Entities>
<Layers>
  <Differences />
  <Missing />
  <New />
</Layers>
<LTypes>
  <Differences />
```

```
    <Missing />
    <New />
  </LTypes>
  <Styles>
    <Differences />
    <Missing />
    <New />
  </Styles>
</CADComparator>
```

A.3 Vergleichsalgorithmus: Methode processLists

```
private Boolean processLists(SortedList listA,
   SortedList listB,
   ComparisonType cType)
{
  int i = 0;
  Boolean exactMatchFound = false;
  float deltaForMatch = MIN_DELTA_EXACT;

  if (cType.Equals(ComparisonType.HandleIdentical))
      deltaForMatch = MIN_DELTA_SIMILAR;

  while (i < listA.Count)
  {
    CADEntity entityA = (CADEntity)listA.GetByIndex(i);

    Boolean advanceCounter = true;
    // we first work on exact matches
    // we do not want to accidentally use a similar match
    // before an exact one
    for (int j = 0; j < listB.Count; j++)
    {
      CADEntity entityB = (CADEntity)listB.GetByIndex(j);

      ComparisonType ct = partsMatch(entityA, entityB,
         deltaForMatch, cType);
      if (cType.Equals(ct) &&
          ComparisonType.Identical.Equals(ct))
      {
        // CADBlock contain the whole Entities..
        // we do not compare them deeply
        if (entityA.Count == 0 ||
            entityA.GetType() == typeof(CADBlock))
        {
          // shrink list by removing same ones
          listB.RemoveAt(j);
          listA.RemoveAt(i);
          advanceCounter = false;
          exactMatchFound = true;
          break;
        }
        else
        {
          Boolean foundOne = false;
          foreach (CADEntity ca in entityA.Entities.AllValues)
          {
```

```
      foundOne = false;
      foreach (CADEntity cb in entityB.Entities.AllValues)
      {
        if (partsMatch(ca, cb, deltaForMatch, cType) ==
            ComparisonType.Identical)
        {
          if (cb.Count > 0)
          {
            // we do not support deeper nesting yet!
            throw new NotImplementedException("Comparison failed.
               Nested structure unexpected! Please report this with
               an example file attached.");
          }
          else
          {
            foundOne = true;
            break;
          }
        }
      }
      // section is not same
      if (!foundOne)
      {
        break;
      }
    }
    if (foundOne)
    {
      // shrink list by removing same ones
      listB.RemoveAt(j);
      listA.RemoveAt(i);
      exactMatchFound = true;
      advanceCounter = false;
      break;
    }
  }
}
// in the second pass Identical matches,
// bec. of lowered exactness can pop up here!
if ((cType.Equals(ComparisonType.HandleIdentical) &&
    ComparisonType.HandleIdentical.Equals(ct))
    ||
    (ComparisonType.Identical.Equals(ct) &&
     cType.Equals(ComparisonType.HandleIdentical)
    ))
{
  CADEntity[] ceDiff = new CADEntity[2];
```

```
            ceDiff[0] = entityA;
            ceDiff[1] = entityB;
            this.itemsDifferent.Add(ceDiff);
            listB.RemoveAt(j);
            listA.RemoveAt(i);
            advanceCounter = false;
            break;
          }
        }

        if (advanceCounter)
            i++;
      }
      return exactMatchFound;
    }
```

Liste von Symbolen und Abkürzungen

Abkürzung	Beschreibung	Vorkommnis
ACL	Access Control List	page 21
CAD	Computer Aided Design	page 7
CVS	Concurrent Versioning System	page 29
DARCS	Verteiltes Version Control System	page 7
DMS	Document Management System	page 15
DWG	AutoCAD Dateiformat. Abkürzung für **D**ra**w**in**g**	page 7
DXF	proprietäre Schnittstelle AutoCAD, ein textbasiertes Austauschformat	page 7
ECM	Enterprise Content Management	page 18
i18n	internationalization	page 31
JCR	Java Content Repository	page 25
PDF	portable document format von Adobe.	page 22
SCM	Software Configuration Management	page 18
SDK	Software Development Kit	page 25
UNO	universal network objects (OpenOffice)	page 28
VCS	Version Control System	page 7
WCM	Web Content Management	page 26

Abbildungsverzeichnis

Tabellenverzeichnis