Grafische Einbindung von Plug-ins im Eclipse Rich Client Platform Umfeld

Praxisbericht

des Studiengangs Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Max Emmert

März 2014

Matrikelnummer, Kurs	8132098, STG-TINF12C
Ausbildungsfirma	Compart AG, Böblingen
Betreuer	Lars Heppler

Sperrvermerk

Die vorliegende Praxisbericht mit dem Titel *Grafische Einbindung von Plug-ins im Eclipse Rich Client Platform Umfeld* ist mit einem Sperrvermerk versehen und wird ausschließlich zu Prüfungszwecken am Studiengang Angewandte Informatik der Dualen Hochschule Baden-Württemberg Stuttgart vorgelegt. Jede Einsichtnahme und Veröffentlichung – auch von Teilen der Arbeit – bedarf der vorherigen Zustimmung durch die Compart AG.

Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meinen Praxisbericht mit dem Thema *Grafische Einbindung von Plug-ins im Eclipse Rich Client Platform Umfeld* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Praxisbericht bei keiner anderen Prüfung vorgelegt habe;

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Stuttgart, März 2014

Max Emmert

Zusammenfassung

Thema der Arbeit
Einbindung von Plug-ins im Eclipse RCP[1]-Umfeld, sowie deren grafische Einbettung in eine bestehende RCP-Anwendung auf Basis von OSGi[2].
Stichworte
OSGi, Eclipse RCP, GUI, Filterprofil, Plug-in
Kurzzusammenfassung
In dieser Arbeit wird untersucht, inwiefern es möglich ist, Plug-ins in eine bestehende Eclipse RCP-Applikation zu integrieren. Dabei steht die grafische Einbindung in das bestehende Softwarekonstrukt im Vordergrund. Die Konzeption und Umsetzung eines Plug-ins im RCP-Umfeld gehört ebenfalls zum Kern dieser Arbeit. Neben den Grundlagen der Eclipse RCP Entwicklung werden verwandte Themen, wie OSGi behandelt.

[1] Rich Client Platform
[2] Open Services Gateway initiative

Summary

Subject Of This Work
Integration of plug-ins in an existing Eclipse RCP environment, as well as their graphical embedding in an RCP application based on OSGi.
Tags
OSGi, Eclipse RCP, GUI, Filterprofil, Plug-in
Executive Summary
In this work it is studied how to integrate plug-ins into an existing Eclipse RCP application. In particular, the graphic integration into the existing software construct is in the foreground. The design and implementation of a plug-in in the RCP environment is also part of the core of this work. Besides the basics of Eclipse RCP development related issues such as OSGi are treated.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Noch vor einigen Jahren war es nicht unüblich, Anwendungen über die Kommandozeile zu steuern. Die funktionalen Aspekte einer Anwendung hatten eine höhere Priorität als ihre Nutzbarkeit. Um komplexe Applikationen über die Kommandozeile zu bedienen ist jedoch oft Expertenwissen notwendig. Deshalb spielt die Entwicklung von grafischen Oberflächen in der heutigen Anwendungsentwicklung eine weitaus größere Rolle. Es reicht oftmals nicht aus, dem Benutzer nur eine grafische Möglichkeit zur Steuerung einer Applikation zu geben. Im Rahmen der Software-Ergonomie spielt die Benutzerführung eine tragende Rolle. So auch bei der Eclipse RCP Applikation DocBridge® Workbench for Mill Plus, einem Programm der Compart AG, die es dem Anwender ermöglicht, Prozesse zur Modifikation und Konvertierung von Dokumenten zu steuern. Der Anwendung DocBridge® Workbench for Mill Plus liegt die Programmiersprache Java zu Grunde. Durch die hohe Verbreitung von Java entstehen immer komplexere Anwendungen. Um die Komplexität solcher Anwendungen handhaben zu können, bedient man sich oft Mitteln zur Modularisierung. Hierfür hat Java jedoch keine eigene Sprachunterstützung. OSGi bietet die Möglichkeit, monolithischen[1] Anwendungen, die den heutigen dynamischen Anforderungen nicht gerecht werden, entgegenzuwirken. Dabei hat sich OSGi von seiner ursprünglichen Anwendung in eingebetteten Systemen emanzipiert und wird heute auf verschiedenste Art verwendet. Anwendungsbereiche sind Server-und Webapplikationen, Mobilfunkgeräte oder Client-Anwendungen wie beispielsweise die Eclipse IDE[2] und ihre RCP-Architektur. Das Eclipse RCP-Framework bietet Möglichkeiten, Client-Anwendungen nach dem Baukastenprinzip zu entwickeln. Dies macht es für die Rich-Client-Entwicklung besonders geeignet. Applikationen, die auf dem Eclipse RCP-Framework basieren, lassen sich durch Plug-ins um Funktionalitäten erweitern.

[1] nicht zerlegbar
[2] Integrierte Entwicklungsumgebung

1.2 Zielsetzung

In der vorausgegangenen Praxisphase wurde vom Studenten ein Programm entwickelt, das es ermöglicht, Filterprofile[1] anhand eines gegebenen XML-Schemas zu aktualisieren. Auf die Funktion der Filterprofile und deren Aktualisierung wird im Kapitel 2 eingegangen. Diese Funktionalität soll nun in die DocBridge® Workbench for Mill Plus integriert werden. Dies würde Kunden bei der Auslieferung eines neuen XML-Schemas die Möglichkeit geben, die Konfiguration ihrer alten Profile beizubehalten beziehungsweise ein neues Filterprofil zu erstellen, das die Konfiguration des alten Filterprofils enthält, jedoch zum neuen XML-Schema valide ist. Darüber hinaus soll es dem Anwender möglich sein, bestehende Profile von ihrem Dateisystem in die DocBridge® Workbench for Mill Plus zu importieren, damit diese zur Konfiguration der In- und Output-Filter verwendet werden können.

1.3 Aufbau der Arbeit

Die Einleitung soll dem Leser einen Überblick über das Thema verschaffen. Es werden sowohl die Motivation für diese Arbeit, als auch deren Zielsetzung, dargestellt. Im Kapitel Grundlagen werden Begriffe und Systeme erklärt, die für das Verständnis und die Nachvollziehbarkeit dieser Arbeit grundlegend sind. Im Hauptteil wird die Vorgehensweise zur Problemlösung detailliert beschrieben. Das Kapitel besteht aus den Abschnitten Analyse, Konzept, Implementierung, Ergebnisse und Ausblick. In der Analyse werden die zugrunde liegenden Probleme analysiert. In dem Abschnitt Konzept wird eine Lösung der behandelten Aufgabe erörtert. Die Umsetzung wird im Abschnitt Implementierung beschrieben. Im Abschluss daran wird auf die Ergebnisse der Arbeit eingegangen. Die Ergebnisse werden zusammengefasst und bewertet. Zum Schluss wird ein Ausblick auf das, im Anschluss an diese Arbeit geplante Vorgehen, gegeben.

[1] Dateien im XML-Format, mit denen Konvertierungsfilter konfiguriert werden

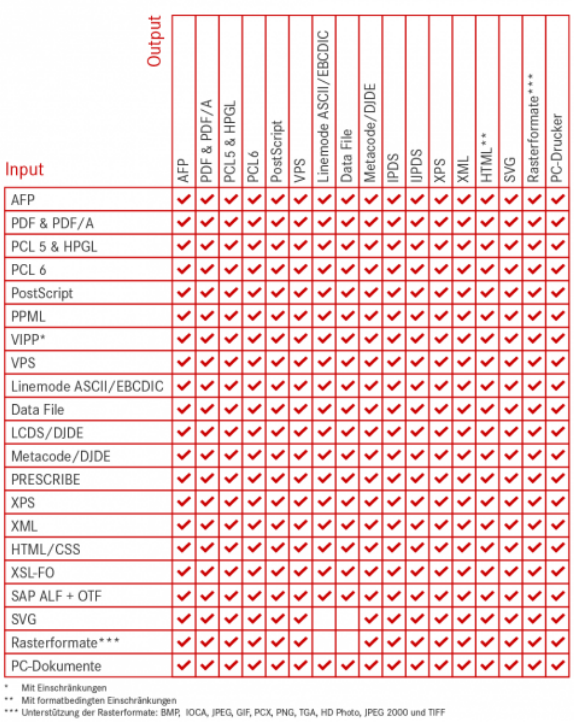
2 Grundlagen

In diesem Kapitel werden grundlegende Technologien und Systeme erklärt, welche im Rahmen dieser Arbeit Verwendung finden.

2.1 DocBridge® Mill Plus

DocBridge® Mill Plus ist eine plattformunabhängige, skalierbare Software für die Anzeige und Verarbeitung von Datenströmen. Sie ermöglicht die Analyse und Modifikation von Dokumenten in verschiedensten Formaten und die Konvertierung in unterschiedliche Ausgabeformate. Abbildung 2.1 zeigt die unterstützten Formate. Die Dokumente lassen sich auf nahezu allen gängigen physikalischen und digitalen Kanälen darstellen. Mit DocBridge® Mill Plus ist der Anwender auch dazu in der Lage, Prozesse die im Dokumentenmanagement üblich sind[1], abzubilden. Der modulare Aufbau der Software lässt es zu, sie mit zusätzlichen Modulen um Ein- und Ausgabeformate sowie Workflows zu erweitern.

[1] Beispielsweise die Modifikation oder das Konvertieren von Dokumenten



Input \ Output	AFP	PDF & PDF/A	PCL5 & HPGL	PCL6	PostScript	VPS	Linemode ASCII/EBCDIC	Data File	Metacode/DJDE	IPDS	IJPDS	XPS	XML	HTML**	SVG	Rasterformate***	PC-Drucker
AFP	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PDF & PDF/A	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PCL 5 & HPGL	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PCL 6	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PostScript	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PPML	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
VIPP*	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
VPS	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Linemode ASCII/EBCDIC	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Data File	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
LCDS/DJDE	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Metacode/DJDE	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
PRESCRIBE	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
XPS	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
XML	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
HTML/CSS	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
XSL-FO	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
SAP ALF + OTF	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
SVG	✔	✔	✔	✔	✔	✔			✔	✔	✔	✔	✔	✔	✔	✔	✔
Rasterformate***	✔	✔	✔	✔	✔	✔			✔	✔	✔	✔	✔	✔	✔	✔	✔
PC-Dokumente	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

* Mit Einschränkungen
** Mit formatbedingten Einschränkungen
*** Unterstützung der Rasterformate: BMP, IOCA, JPEG, GIF, PCX, PNG, TGA, HD Photo, JPEG 2000 und TIFF

Abbildung 2.1: Mögliche Verarbeitungsformate der DocBridge® Mill Plus

2.2 DocBridge® Workbench for Mill Plus

DocBridge® Workbench for Mill Plus bietet eine grafische Benutzeroberfläche für die Funktionalitäten der DocBridge® Mill Plus wie Filterkonfigurationen und die Formatunabhängige Dokumentendarstellung. In der DBWB[1] können einzelne, mehrere oder in Stapeln zusammengefasste Dokumente angezeigt werden. Es werden alle in Abbildung 2.1 aufgeführten Formate unterstützt. Geladene Dokumente können in der in Abbildung 2.3 vorgestellten Dokumentenperspektive einheitlich dargestellt werden. Alle Dokumente, die in einem der unterstützten Formaten vorliegen, können im Anzeigebereich untersucht werden. So ist es dem Nutzer möglich, sich unabhängig vom Format des Dokuments, völlig auf dessen Inhalt zu konzentrieren. In der Prozessperspektive (Abbildung 2.2) können Verarbeitungsschritte konfiguriert, validiert und lokal ausgeführt werden. Die erstellte Prozesskonfiguration kann exportiert und in der DocBridge® Mill Plus ausgeführt werden. Die Konfiguration der Prozesse erfolgt überwiegend über Drag & Drop. Die Prozessperspektive ist die Standardperspektive der DBWB.

[1] DockBridge Workbench

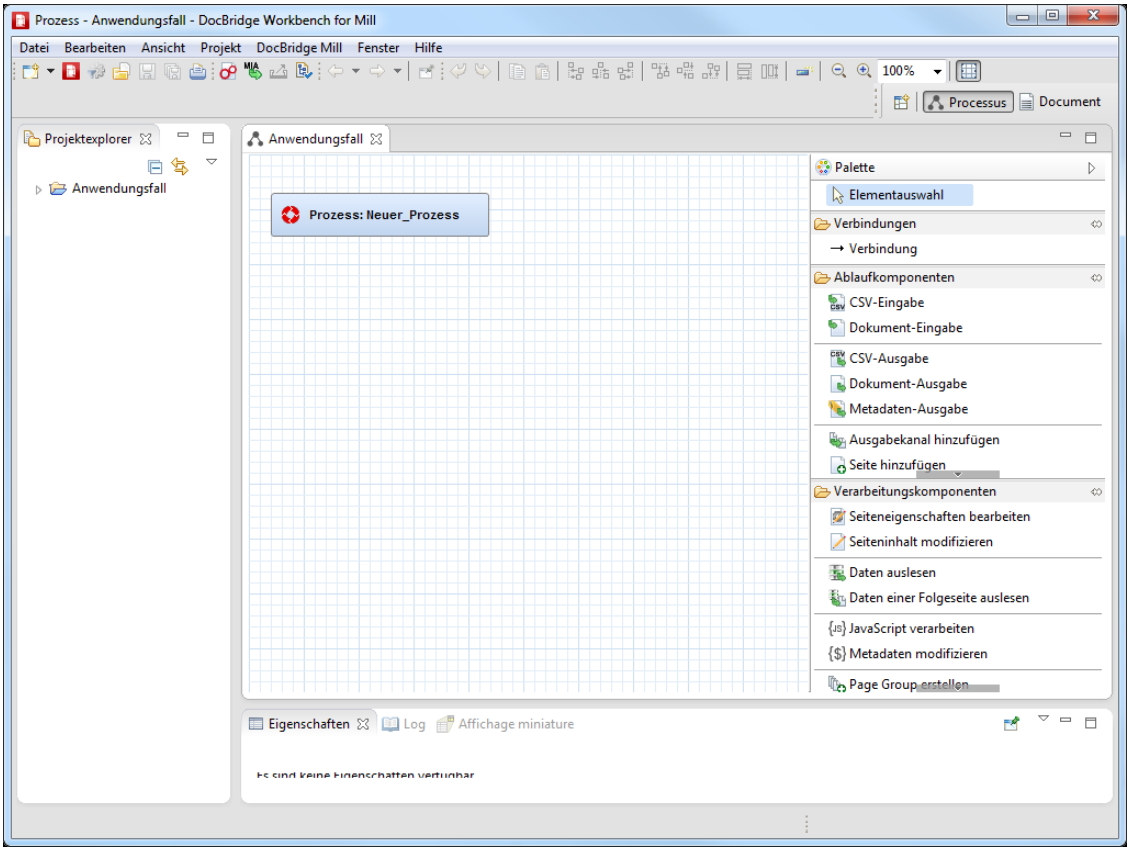


Abbildung 2.2: Prozessperspektive

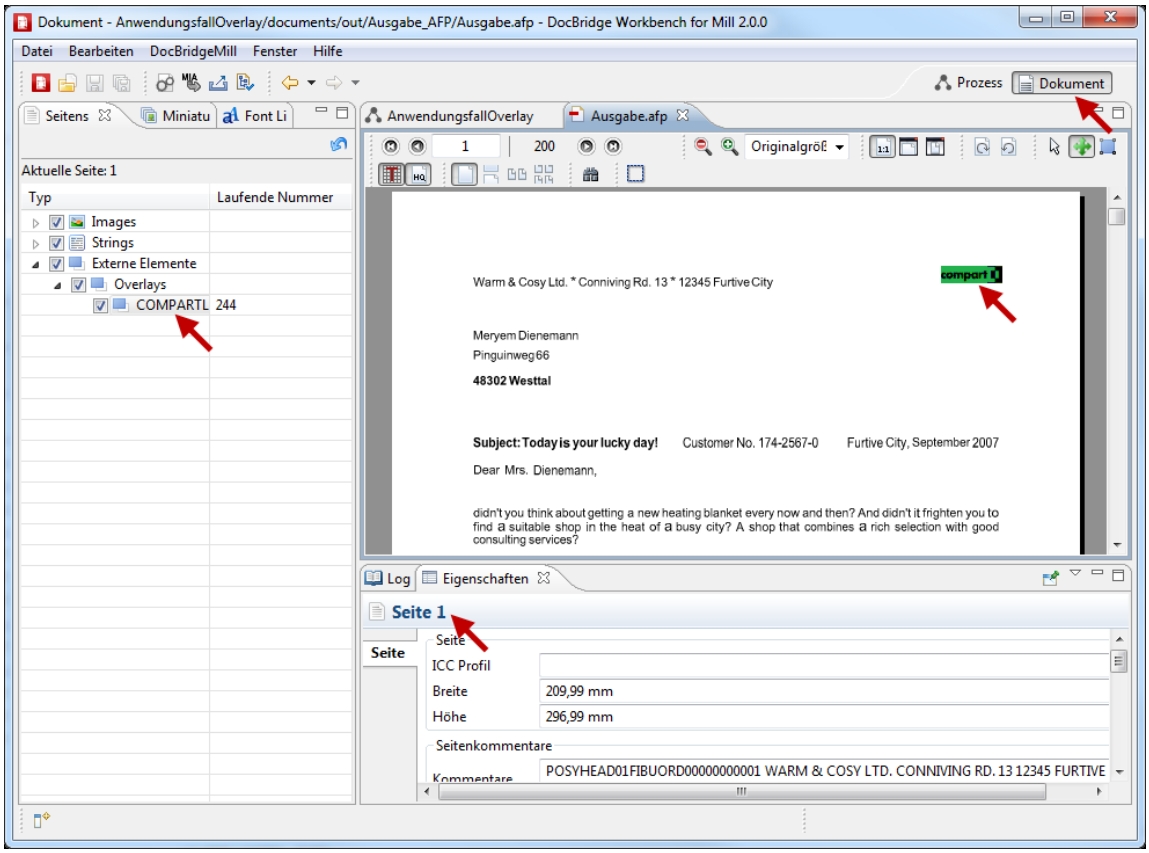


Abbildung 2.3: Dokumentperspektive

2.3 OSGi

Java stellt nur begrenzte Möglichkeiten zur Modularisierung bereit. „Mit Modularisierung bezeichnet man die Aufteilung eines Softwaresystems in überschaubare Einzelteile, die jeweils einen abgeschlossenen Aufgabenbereich umfassen und untereinander mittels wohldefinierter Schnittstellen miteinander verbunden sind.“ (Reichert, 2009, S.11) Java-Klassen lassen sich zwar unter der Verwendung von Packages gruppieren, jedoch ist es nicht möglich diese Gruppierungen und Restriktionen zur Laufzeit beizubehalten(Ebert, 2004, vgl. S.60). Die Möglichkeit, einzelne Klassen als `package-protected` zu deklarieren ist „für die Strukturierung und Abgrenzung von Softwaremodulen meist unzureichend“(Ebert, 2004, S.60). OSGi ermöglicht die Verwaltung der Sichtbarkeit von Packages zur Laufzeit. OSGi ist ein Framework, welches sich auch dem Problem der Modularisierung annimmt und die Entwicklung von modularen Anwendungen auf der Java-Plattform ermöglicht. Abbildung 2.4 zeigt das OSGi-Schichtenmodell, das auf der JRE[1] aufsetzt. OSGi-Anwendungen bestehen aus einzelnen Modulen, die auch Bundles genannt werden. Im Eclipse-Umfeld wird auch der Begriff „Plug-in“ bedeutungsgleich für Bundles verwendet. Ein Bundle ist ein JAR-Archiv, das aus der Beschreibung des Bundles[2], Java-Klassen und Ressourcen besteht. Jedes Bundle hat einen eigenen Lebenszyklus. Dies bedeutet, dass zur Laufzeit einzelne Bundles dynamisch geladen und entfernt werden können. Die Spezifikation von OSGi wird von der OSGi Alliance erstellt. Die OSGi Alliance wurde 1999 von rund 30 namhaften Firmen gegründet, um den Anforderungen im Bereich der eingebetteten Systeme gerecht zu werden (Kriens, 2008, vgl. S.3). Die Eclipse IDE arbeitet seit Version 3.0 mit der OSGi Implementierung Equinox. Bei OSGi Services handelt es sich um Java-Objekte, die unter einem Interface in der OSGi Service-Registry angemeldet werden. In der Service-Registry angemeldete Services können von anderen Bundles verwendet werden. Services können genau wie OSGi-Bundles dynamisch geladen und entfernt werden. Wird ein Bundle, das einen Service bereitstellt gestoppt, so muss das verwendende Bundle auf das Wegfallen des Service entsprechend reagieren (OSGiAlliance, 2012, vgl. S.114).

[1] Java Runtime Environment
[2] Manifest.mf Datei

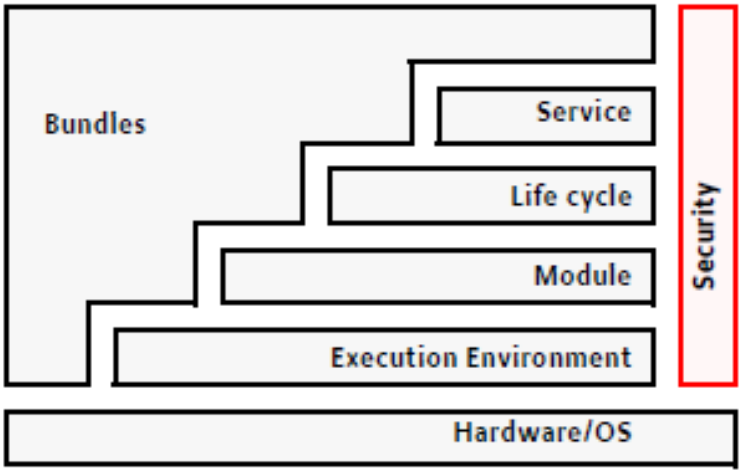


Abbildung 2.4: OSGi Schichtenmodell (OSGiAlliance, 2012, aus S.2)

2.4 SWT

Das Standard Widget Toolkit stellt das Framework für grafische Oberflächen auf der Eclipse-Plattform bereit. Diese grafischen Oberflächen sind aus Widgets aufgebaut. „Traditionally, a widget is thought of as an abstract device that is useful for a particular purpose.“(Steve Northover, 2004, Kap.1) SWT[1] stellt eine Java-Schnittstelle für die nativen GUI-Bibliotheken verschiedener Betriebssysteme zur Verfügung. Hierzu gehören Microsoft Windows, Linux[2] und Mac OS X[3]. Da auf die spezifischen Steuerelemente des jeweiligen Betriebssystems zugegriffen wird, kann mit SWT auf den oben genannten Plattformen ein natives Aussehen und Verhalten erzielt werden, wie in Abbildung 2.5 veranschaulicht wird.

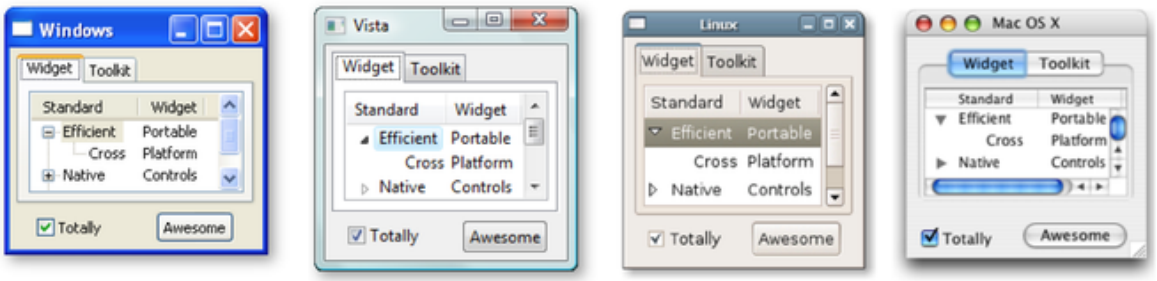


Abbildung 2.5: Darstellung von SWT-Widgets auf verschiedenen Betriebssystemen (Ebert, 2004)

1 Standart Widget Toolkit
2 GIMP Toolkit
3 Cocoa

2.5 Eclipse RCP

Eclipse RCP ist eine Plattform zur Entwicklung von Desktop-Anwendungen. Sie entstand aus der Eclipse IDE, die sich aufgrund ihrer allgemeinen Natur anbot. Das Eclipse RCP-Grundgerüst kann durch eigene Anwendungsfunktionalitäten erweitert werden. Viele der vorhandenen Komponenten können in eigenen Anwendungen genutzt werden. Als Beispiel wären hier Komponenten für eine Hilfe-Funktion oder das erweiterbare Plug-in-System zu nennen. Vorteile der Anwendungsentwicklung mit Eclipse RCP sind die bereits vorhandenen Grundstrukturen für den Aufbau von GUI-Anwendungen und der modulare Aufbau. Dieser ermöglicht es, dass Erweiterungen auf Eclipse RCP-Basis miteinander harmonieren. Die zur Implementierung benötigten Werkzeuge sind bereits in der Eclipse IDE[1] enthalten.

2.6 Target Platform

Eclipse RCP-Applikationen werden für eine konfigurierbare Zielplattform[2] entwickelt. Alle externen Abhängigkeiten werden über die Target Platform geladen. Soll mit Plug-ins die Eclipse IDE erweitert werden, verwendet man standardmäßig die Eclipse IDE selbst als Target Platform (Ebert, 2004, vgl. S.14). Bei der Entwicklung einer eigenen RCP-Anwendung, oder Plug-ins für eine solche, ist das Verwenden der Eclipse IDE „unvorteilhaft, da die Eclipse-Installationen meist zu umfangreich sind und sich bei den einzelnen Entwicklern stark unterscheiden“(*Eclipse Magazin 3.10*, 2010, S.13). In einem solchen Fall ist es günstiger, eine eigene Target Platform zu definieren. Durch die Definition der Target Platform kann festgelegt werden, in welcher Version der Applikation bestimmte Plug-ins eingebunden werden. Im Idealfall arbeiten alle Entwickler eines Projekts mit der selben Target Platform. So wird eine einheitliche Entwicklungsumgebung sichergestellt und Versionskonflikte vermieden.

[1] Eclipse for RCP/Plug-in Developers
[2] Target Platform

2.7 Apache Maven

Das Tool Maven „ist ein weiteres Top-Level-Projekt der ASE[1]. Sein Funktionsumfang ist etwas weiter gefasst als der von vergleichbaren Werkzeugen wie beispielsweise Ant, im Kern handelt es sich bei Maven aber um ein Werkzeug zur Durchführung eines Build-Prozesses“(Popp, 2013, S.75). Apache Maven basiert selbst auf Java und bietet die Möglichkeit, Java Programme standardisiert zu erstellen und zu verwalten. Mit Maven wird versucht, das Softwaredesign-Paradigma "Konvention vor Konfiguration"[2] für den gesamten Lebenszyklus der Softwareerstellung abzubilden. Dabei werden Schritte wie das Kompilieren und Testen so weit wie möglich automatisiert. Orientiert man sich bei der Entwicklung an den Maven Standards, ist es nur an wenigen Stellen nötig, Konfigurationen vorzunehmen.

2.8 SonarQube™

Diese Plattform für statische Codeanalyse wird zur Sicherung der Codequalität eingesetzt. SonarQube™ selbst ist wie Maven in Java implementiert, unterstützt jedoch eine Vielzahl von Programmiersprachen. Neben Java werden unter anderem die Sprachen C#, C/C++, PL/SQL und Cobol unterstützt. SonarQube™ analysiert Sourcecode auf Fehler oder unsaubere Programmierung und protokolliert die Ergebnisse in eine Datenbank. Auf einer Webseite werden die Ergebnisse der Analyse dargestellt. SonarQube™ wird auch als Eclipse-Plug-in ausgeliefert. Dieses Plug-in kann Java Projekte mit Projekten aus der SonarQube™ Datenbank assoziieren. Es ermöglicht eine lokale Nutzung und somit ein schnelleres Feedback für den Entwickler. Plattformen wie SonarQube™ sind von großer Bedeutung, um die Codequalität in großen Softwareprojekten zu gewährleisten.

2.9 Plug-in zur Profilkonvertierung

Vom Studierenden wurde während der letzten Praxiseinheit ein Plug-in im OSGi-Umfeld entwickelt, welches dazu verwendet werden kann Filterprofile zu aktualisieren. Die Konfiguration von Konvertierungsabläufen wird in Filterprofilen festgehalten.

[1] Apache Software Foundation
[2] Reduktion der Komplexität von Konfigurationen

Deren Aufbau ist in einem XML-Schema definiert. Kommen Funtionalitäten oder Komponenten bei der Filterkonfiguration hinzu, wird das Schema aktualisiert und somit abgeändert. Bestehende Filterprofile sind nun nicht mehr zum Schema valide. Das Plug-in zur Profilkonvertierung liest die Informationen aus dem alten Profil mit Hilfe von verschiedenen Java API[1]s aus und generiert anhand des Schemas ein neues Profil, das zum Schema valide ist. Sämtliche Inhalte werden dabei migriert. Es kommen vor allem die APIs Jsoup[2] und JaxB[3][4] zur Verwendung.

[1] Application Programming Interface
[2] API zum Arbeiten mit degeneriertem HTML
[3] Java Architecture for XML Binding
[4] API für das Erstellen von Klassen aus einer XML-Schema-Instanz

3 Hauptteil

3.1 Analyse

In diesem Abschnitt werden die Anwendungsfälle nicht funktionalen Anforderungen dargestellt der zu entwickelnden Software zusammengefasst.

3.1.1 Anwendungsfälle

Die verschiedenen Anwendungsfälle für die zu entwickelnde Grafische Oberfläche werden in diesem Abschnitt erläutert.

Profil importieren

Der Kunde soll in der Lage sein, ein Profil in seinem Datenspeicher über die Menüführung der Workbench for Mill Plus auszuwählen. Das ausgewählte Profil soll in der Workbench geöffnet und in einem neuen Editor zur weiteren Bearbeitung angezeigt werden.

Profil aktualisieren

Ist ein Profil im Editor der Workbench for Mill Plus geöffnet, soll es dem Anwender über verschiedene Methoden der Benutzerführung ermöglicht werden, die Aktualisierung des Profils zu starten. Ist das Profil zum aktuellen XML-Schema valide, soll keine Aktualisierung durchgeführt werden.

Importiertes Profil verwenden

Ein importiertes Profil soll nach seinem Import in der DBWB im Workflow des Benutzers verwendet werden können.

Erzeugtes Profil verwenden

Ein vom Aktualisierungsvorgang erzeugtes Profil soll nach seiner Erstellung in der DBWB im Workflow des Benutzers zur Filterkonfiguration verwendet werden können.

3.1.2 Nicht funktionale Anforderungen

Nicht funktionale Anforderungen sind Anforderungen, an die "Qualitätïn welcher die geforderte Funktionalität zu erbringen ist. In diesem Abschnitt werden die nicht funktionalen Anforderungen erklärt.

Handhabung

Bei den Anwendern der DBWB handelt es sich meist um Personen, die im Dokumentmanagement tätig sind. Von Ihnen kann kein programmiertechnisches Expertenwissen erwartet werden, da dies nicht zum Schwerpunkt ihrer beruflichen Aufgaben gehört. Somit ist es erforderlich, die Funktionalität zur Aktualisierung von Profilen möglichst intuitiv in den bestehenden Workflow zu integrieren.

Wartbarkeit

Die einzelnen OSGi-Bundles sowie deren Zusammenspiel müssen einfach wartbar sein, damit sie bei der Integration in das bestehende Produkt keine Mehrarbeit verursachen können. Auch für Updates und zukünftige Entwicklungen ist eine gute Wartbarkeit von großer Bedeutung. Ist eine Software schlecht wartbar, lassen sich neue oder geänderte Anforderungen meist nur mit hohem Aufwand umsetzen.

Portierbarkeit

Das zu entwickelnde Plug-in soll auf allen Systemen verfügbar sein, auf denen auch die DBWB ausgeliefert wird. Die Portierbarkeit bei modular aufgebauten Systemen hat eine große Bedeutung. Wird ihr wenig Aufmerksamkeit geschenkt, kann es sein, dass das entwickelte System nicht auf allen Zielplattformen das gleiche Verhalten aufweist.

Konsistenz

Die Erweiterungen der DBWB sollen eine Konsistenz in der Benutzerführung aufweisen. Zusätzliche Module und Plug-ins sollen nahtlos in den bestehenden Arbeitsablauf integriert werden, um die gewohnte Benutzbarkeit für den Anwender zu gewährleisten.

3.2 Konzept

In diesem Abschnitt wird das, dem zu entwickelnden Plug-in zugrundeliegende, Konzept erläutert.

3.2.1 Grafische Einbettung

Die neu bereitgestellten Funktionalitäten sollen sich nahtlos in die bestehende Anwendung DocBridge® Workbench for Mill integrieren. Darüber hinaus soll die grafische Einbettung den Standards der Compart AG für GUI[1]s genügen. Hierzu wurden die Benutzeroberfläche der DBWB und firmeninterne Designvorlagen untersucht. Um eine hohe Benutzbarkeit zu gewährleisten werden dem Anwender mehrere Möglichkeiten gegeben, auf die hinzugefügten Funktionalitäten zuzugreifen. Um die grafische Einbindung zu visualisieren und zu veranschaulichen wurden Mockups, funktionsunfähige Modelle, angefertigt.

[1] Graphical User Interface

3.2.2 Prototyp

Um die in den Anforderungen geforderten Funktionalitäten herzustellen werden diese erst in einem Prototypen simuliert. Dieser beinhaltet eine Komponente zum importieren und eine Komponente zum aktualisieren von Filterprofilen. Der Prototyp konzentriert sich bei der Aktualisierungskomponente zunächst auf den Dokumenttyp AFP[1].

Komponente für den Profilimport

Filterprofile werden in der DBWB in einem Profile Repository verwaltet. Für gewöhnlich wird mit jeder neuen Version eines XML-Schemas ein default-Profil mitgeliefert, anhand dessen Vorlage sich der Anwender ein eigenes Profil erstellen lassen kann. Das erstellte Filterprofil kann beliebig modifiziert werden. Das default-Profil ist nicht zur Modifikation vorgesehen. Die Komponente liest aus dem importierten Profil die Version aus. Die Version ist in jedem Profil als Attribut abgelegt. Anhand der Version und einem eindeutigen Identifier wird eine neue Version im Profile Repository angelegt. Das Profil wird als neues default-Profil dieser Version abgelegt. Das hat den Vorteil, dass der Anwender sein importiertes Profil in der Dokumentenperspektive der DBWB untersuchen, jedoch nicht verändern kann. Soll das importierte Profil modifiziert werden, muss der Benutzer anhand des neuen default-Profils ein neues Profil generieren lassen. Große Bedeutung kommt auch der Benutzerführung zu. In jeder der beiden Perspektiven der DBWB ist der Import auf unterschiedliche Art und Weise zu handhaben.

Prozessperspektive

Dem Anwender steht in dieser Perspektive eine Projektnavigation zur Verfügung. Er soll beim Import eine Auswahlmöglichkeit für das Zielprojekt haben. Importierte Profile werden in einem Unterordner `profiles` im ausgewählten Projekt abgelegt. Anfangs war es vorgesehen, in der Prozessperspektive mehrere Filterprofile gleichzeitig importieren zu können. Während der Implementierung der Importkomponente wurde klar, dass dies der Konsistenz und Nutzerführung nicht zuträglich ist. Die Funktionalität wurde deshalb auf den Import eines einzelnen Profils beschränkt. Nach dem Import wird das Profil im Profile Repository abgelegt und in einem Editor geöffnet. Existiert im Projektunterverzeichnis `profiles` bereits ein Profil für den entsprechenden Filtertyp

[1] Apple Filing Protocol

soll der Nutzer die Möglichkeit haben, das existente Profil zu überschreiben, das zu importierende Profil umzubenennen oder seine Aktion abzubrechen.

Dokumentenperspektive

In der Dokumentenperspektive betrachtet der Benutzer oft nur ein einzelnes Dokument, das mit Hilfe verschiedener Filter konvertiert werden kann. Deshalb ist es unnötig mehrere Profile importieren zu können. Die Komponente zur Projektnavigation ist in dieser Perspektive ebenfalls deaktiviert. Somit soll es beim Import in der Dokumentenperspektive auch nicht möglich sein, ein Projekt auszuwählen, in dem das Profil gespeichert wird. Es wird nur im Profile Repository abgelegt und in einem Editor geöffnet.

Komponente zur Aktualisierung von Profilen

Bei der Aktualisierung von Profilen soll ein bereits vorhandenes Plug-in verwendet werden, das die Funktionalität der Konvertierung von Profilen zur Verfügung stellt. Den Use Case zeigt Abbildung 3.1. Geplant ist, dass für jeden Filtertyp ein eigenes Plug-in erstellt wird. Diese werden bei Bedarf eingebunden. Wie auch bei der Importkomponente soll sich die Aktualisierungskomponente in den bestehenden Arbeitsfluss integrieren. Ist ein Filterprofil in einem Editor ausgewählt, hat der Anwender verschiedene Möglichkeiten den Konvertierungsprozess anzustoßen. Neben einem Eintrag im Hauptmenü soll die Aktion auch über eine Toolbar und ein Kontextmenü verfügbar sein. Nach starten des Prozesses über eine der genannten Möglichkeiten wird das aktuell geöffnete Profil ermittelt. Anhand des root-Tags[1] wird der Filtertyp(z.B. AFP,PDF[2]) ermittelt. Anhand dieses Filtertyps wird das benötigte Konvertierungs-Plug-in festgestellt und dessen convert()-Methode mit dem ausgewählten Profil als Übergabeparameter gestartet. Sobald die Konvertierung abgeschlossen ist, wird dem Anwender das originale und das generierte Profil angezeigt, um weitere Modifikationen vornehmen zu können, wie im Sequenzdiagramm Abbildung 3.2 dargestellt ist. In einem Vergleichseditor soll der Nutzer darüber entscheiden, welche Änderungen übernommen und welche verworfen werden sollen. Die nachfolgenden Unterabschnitte stellen drei Ansätze dar, wie ein solcher Vergleichseditor realisiert werden kann.

[1] Hauptelement in der Profildatei
[2] Portable Document Format

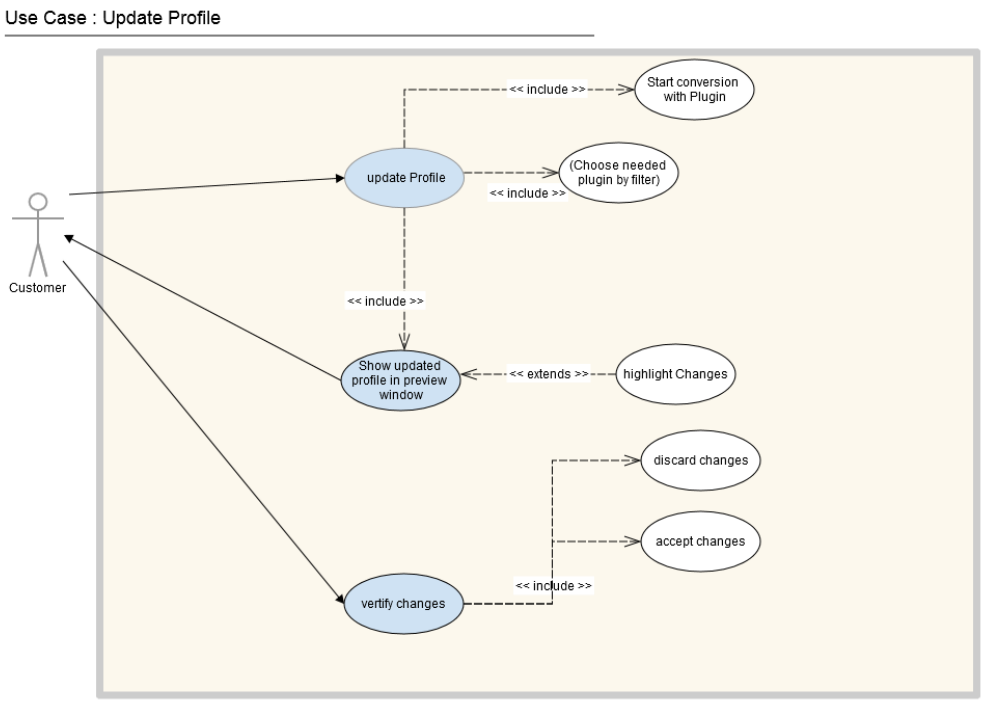


Abbildung 3.1: Use Case Diagramm zum Aktualisieren der Profile

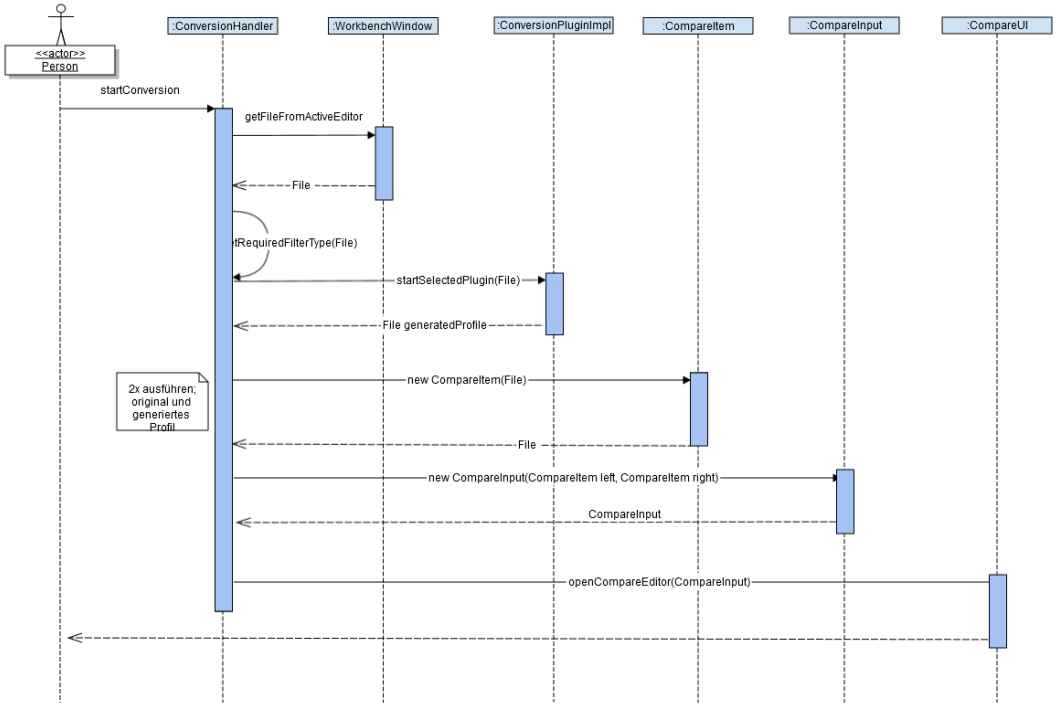


Abbildung 3.2: Sequenzdiagramm zur Aktualisierungskomponente

Erweiterung des Dokumenteneditors

Der Editor der DBWB ermöglicht es, eine Vorschau von zu konvertierenden Dokumenten anzuzeigen. Ist beispielsweise ein Dokument im AFP-Format gegeben und soll nach PDF konvertiert werden, so lässt sich das Ergebnis in einem Editor bereits zur Laufzeit betrachten. Geplant war das Überschreiben dieses Editors mit einem Standard XML-Editor. Dieser Ansatz musste verworfen werden, da dies im konzeptionellen Widerspruch zum etablierten Dokumenteneditor stand.

Erstellen eines spezialisierten Editors

Der nächste Ansatz war es, einen Editor speziell zum Vergleich von zwei Filterprofildateien zu entwickeln. Ein EditorPart soll zwei weitere EditorParts enthalten, in denen das originale und das konvertierte Profil angezeigt werden. Eclipse RCP sieht jedoch vor, dass ein Editor in Beziehung zu genau einer Datei steht. Deshalb musste dieser Ansatz ebenfalls verworfen werden.

Nutzung von CompareUI[1]

Die Java Klasse `org.eclipse.compare.CompareUI` bietet einen Einstiegspunkt, um konfigurierbare Vergleichsoperationen auf beliebige Ressourcen durchzuführen. Das Ergebnis eines Vergleichs kann in einem speziellen Vergleichseditor geöffnet werden. Dieser Editor ermöglicht das dynamische Vergleichen einzelner Elemente von Dokumenten, was besonders am Beispiel von XML[2] oder HTML[3] ersichtlich wird. Auch das Eclipse Plug-in eGit[4] nutzt die Klasse CompareUI, um Dateien in der Versionierungshistorie zu vergleichen. Ein solcher Vergleich ist in Abbildung 3.3 zu sehen. Da die Vorteile dieser Technologie nicht von der Hand zu weisen sind, wird dieser Ansatz zur Entwicklung zur Darstellung des Ergebnisses des Konvertierungs-Plug-ins gewählt.

3.2.3 Design der Bundles

In der Softwareentwicklung ist die Unterteilung der Software in kleinere Einheiten vorteilhaft, da dies den Test, die Wartung und die Entwicklung allgemein vereinfacht.

[1] Java API um Vergleich von Objekten
[2] Extensible Markup Language
[3] Hypertext Markup Language
[4] Plug-in zur grafischen Bedienung der Versionierungssoftware Git

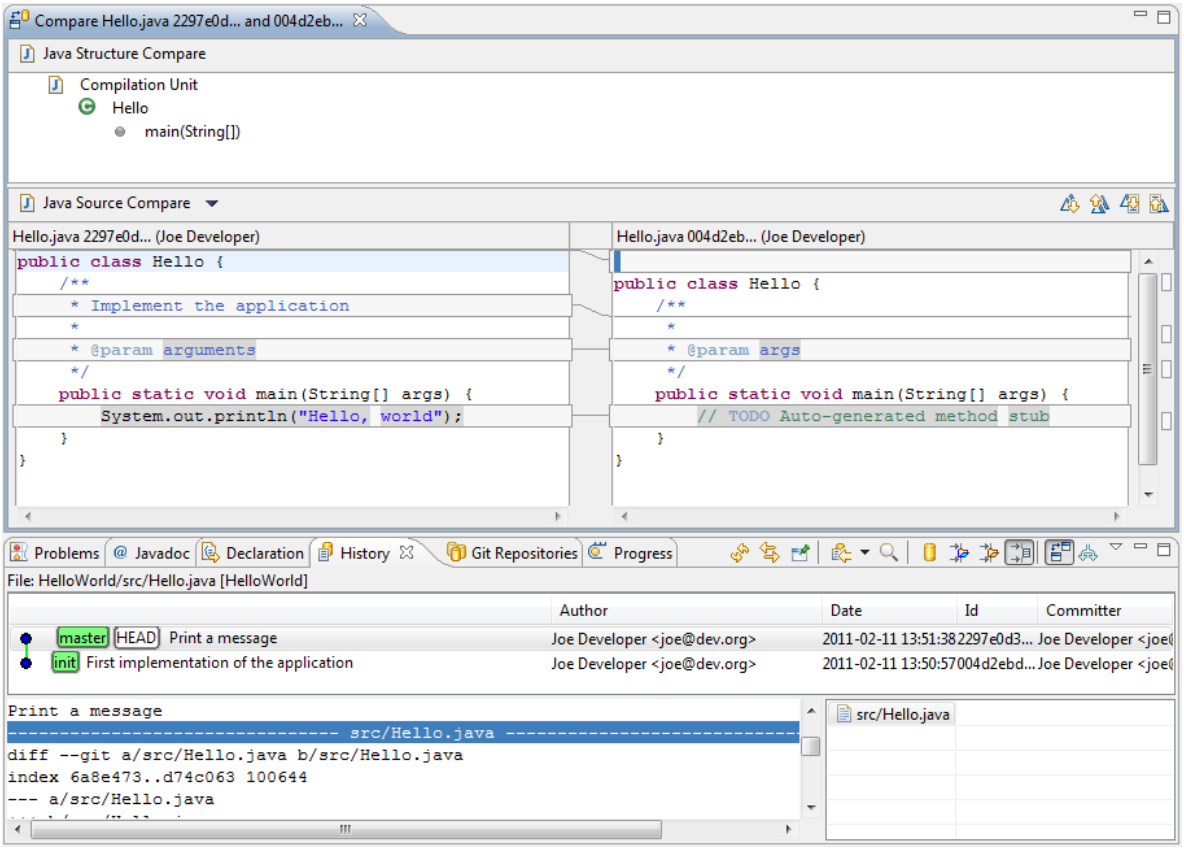


Abbildung 3.3: Verwendung der API CompareUI am Beispiel von eGit

3.3 Implementierung

Dieser Abschnitt beschreibt die Implementierung der Komponenten für den Profilimport und die Aktualisierung der Filterprofile.

3.3.1 Entwicklungsumgebung

Die Implementierung der Bundles erfolgt in der IDE Eclipse[1]. Da RCP auf dem Eclipse-Framework basiert, liegt die Wahl dieser IDE nahe. Für das Konfigurationsmanagement wird das Build-Management-Tool Apache Maven 2 verwendet, das in Abschnitt 2.7 beschrieben wird. Als Laufzeitumgebung wird die JRE6 verwenden, da diese auch bei den bestehenden Bundles im Entwicklungsumfeld zum Einsatz kommt. Das Java-Projekt wird automatisiert täglich auf einem Jenkins[2] System gebaut. Um die Codequalität des Projekts zu gewährleisten, wird Sonarqube™ eingesetzt. Um eine gute, standardisierte Codequalität zu gewährleisten, wird SonarQube™ eingesetzt (siehe Abschnitt 2.8). Zur Versionsverwaltung wird die Versionierungssoftware Git, die ursprünglich für die Quellcode-Verwaltung des Linux-Kernels entwickelt wurde, genutzt. Die grafische Git-Repository-Management-Lösung Atlassian Stash ermöglicht ein übersichtliches Verwalten der genutzten Repositories.

3.3.2 Komponente für den Profilimport

Extensions

Unter Verwendung des Extension-Point-Mechanismus von Eclipse RCP wurden in der Plugin.xml des Bundles verschiedene Extensions angelegt.

Grafische Steuerelemente

Für das ergänzen der Benutzeroberfläche der DBWB wird der Extension-Point

`org.eclipse.ui.menu` verwendet. Sowohl in der Toolbar der DBWB als auch in ihrer Menüleiste werden Buttons bzw Menüeinträge hinzugefügt. Das aktivieren eines solchen Menüeintrags oder Buttons startet den in Abschnitt 3.3.2 beschriebenen

[1] Version Kepler
[2] System zur kontinuierlichen Integration

Handler. Listing 3.1 zeigt den Einsatz einer Extension, um eine neue Menükontribution anzulegen.

Handler

Neben den Extensions für die grafischen Steuerelemente wird eine Extension am Extension-Point `org.eclipse.ui.handlers` registriert. Dieser ImportHandler ist aktiv, falls in der Prozessperspektive ein Projekt selektiert ist, oder der Nutzer sich in der Dokumentenperspektive befindet (vgl. Abschnitt 3.2.2).

Listing 3.1: Exemplarische Extension Points für Toolbareintrag und Importhandler

```
1  /*
2   * Extension Point fuer einen Toolbareintrag
3  */
4     <extension point="org.eclipse.ui.menus">
5                  <menuContribution
6              allPopups="false"
7              locationURI="toolbar:org.eclipse.ui.main.toolbar?
8              after=net.compart.dbwb.filterconfigurator.
9              documenteditor.exportProfilesButton">
10           <toolbar
11              id="net.compart.dbwb.filterconfigurator.toolbar">
12              <command
13                    commandId="com.compart.profileconverter.
14                    ui.importmenu"
15                    icon="icons/import_wiz.gif"
16                    id="com.compart.profileconverter.ui.
17                    importmenu.toolbar.main"
18                    label="%importprofile.label"
19                    style="push"
20                    tooltip="%importprofile.tooltip">
21              </command>
22           </toolbar>
23        </menuContribution>
24        </extension>
25 /*
26  * Extension Point fuer den Import Handler
27 */
28        <extension point="org.eclipse.ui.handlers">
29              <handler
30                 class="com.compart.profileconverter.
31                 ui.importmenu.internal.ImportHandler"
32                 commandId="com.compart.profileconverter.
33                 ui.importmenu">
34                 <enabledWhen>
35                   <or>
```

```
36                  <with
37                        variable="selection">
38                     <iterate
39                           ifEmpty="false"
40                           operator="or">
41                        <instanceof
42                            value="org.eclipse.core.
43                            resources.IProject">
44                        </instanceof>
45                     </iterate>
46                     <count
47                          value="1">
48                     </count>
49                  </with>
50                  <with variable="activeWorkbenchWindow.
51                  activePerspective">
52                      <equals
53                          value="com.compart.dbwb.
54                          productivitytools.
55                          documentPerspective">
56                      </equals>
57                  </with>
58                </or>
59             </enabledWhen>
60          </handler>
61    </extension>
```

Implementierte Klassen

In diesem Abschnitt werden die implementierten Klassen beschrieben. Das implementierte OSGi-Bundle verfügt nicht über eine Aktivator-Klasse, da der default Aktivator in unserem Falle ausreichend ist. Wie in (Daum, 2007, S.70) beschrieben kontrolliert die Aktivator-Klasse den Lebenszyklus des Plug-ins. Bei der Implementierung der Klassen wurden die in Tabelle 3.1 Bibliotheken verwendet.

ImportHandler

In der von `org.eclipse.core.commands.AbstractHandler` abgeleiteten Klasse ImportHandler wird zunächst überprüft, in welcher Perspektive sich der Benutzer beim Aufruf befand. Handelt es sich um die Prozessperspektive, wird mit Hilfe eines Selection-Services, der in Listing 3.2 abgebildet ist, das selektierte Projekt im Projektnavigator ermittelt. Ein FileDialog Widget `org.eclipse.swt.widgets.FileDialog` wird geöffnet, über den der Benutzer das zu importierende Profil auswählen kann.

Existiert kein Filterprofil des gleichen Typs im Projekt, so wird die Datei im `profiles`-Unterverzeichnis des Projekts kopiert und mit Hilfe des ProfileRepositoryLocationProvider im ProfileRepository der DBWB abgelegt. Existiert bereits ein Profil des gleichen Typs wird ein Dialog erzeugt, der dem Benutzer verschiedene Verfahrensmöglichkeiten bietet. Das im Projekt bestehende Profil kann überschrieben werden oder das zu importierende Profil kann umbenannt werden. Mit einem Enumerationsalgorithmus wird ein neuer Name erzeugt.

Listing 3.2: Nutzung eines Selection Services, um das gewähle Projekt zu ermitteln

```
1  private IProject getSelectedProject()
2          {
3          IWorkbenchWindow window =
4                this.page.getWorkbenchWindow();
5          ISelection selection =
6                window.getSelectionService().getSelection(PROJECT_NAV);
7          if (!(selection instanceof IStructuredSelection))
8              {
9              return null;
10             }
11
12         Object o =
13               ((IStructuredSelection)selection).getFirstElement();
14         IProject activeProject = (IProject)o;
15         return activeProject;
16         }
```

ProfileRepositoryLocationProvider

Der ProfileRepositoryLocationProvider wurde von einer in der DBWB existierenden LocationProvider-Klasse abgeleitet. Dieser LocationProvider sucht nach, mit der DocBridge Mill Plus von Haus aus mitgelieferten, Filterprofilen und deren dazugehörigen XML-Schemata. Diese werden bereitgestellt und ins ProfileRepository kopiert. In der Klasse ProfileRepositoryLocationProvider werden viele Funktionen, die zur Ermittlung der Filterprofile dienen überschrieben, um eine eigene Profilversion zu erstellen. Ein Überladen des Konstruktors bietet die Möglichkeit, ein vom bestehenden System unabhängiges Profil einzubringen und ihm eine generierte Version zuzuordnen. Es wird eine eigene Location erstellt, mit der später das ProfileRepository aktualisiert werden kann. Listing 3.3 zeigt den Aufruf des LocationProviders und die Aktualisierung des ProfileRepository.

Listing 3.3: Methode, um Filterprofil im ProfileRepository abzulegen

```
1
```

```
2   private void pushfileToProfileRepository(File profile)
3           {
4           if (profile.exists() && profile.isFile())
5               {
6               ProfileRepositoryLocationProvider locationProvider = new
                    ProfileRepositoryLocationProvider(profile);
7               IRepository profileRepository =
                    ProfileRepository.getProfileRepository();
8               profileRepository.updateLocations(locationProvider);
9               }
10          }
```

Messages

Im Zusammenspiel mit verschiedenen Java-Properties-Dateien[1], wird über die Klasse `Messages` die Lokalisierung der Nachrichten und Labels der RCP-Applikation gesteuert. Dabei werden in der Klasse `Messages` alle benötigten Strings deklariert. Diese werden in Java-Properties-Dateien entsprechend mit der jeweiligen Lokalisierung definiert.

3.3.3 Komponente zur Aktualisierung von Profilen

Da der Aufgabenzeitraum für die Analyse, Konzeption und Implementierung beider Komponenten zu knapp bemessen war, konnte die Aktualisierungskomponente nicht vollständig implementiert werden. Es existiert jedoch ein Prototyp, der mit der Technologie CompareUI(Abbildung 3.3), die bereits im Konzept erwähnt wird, realisiert wurde. Es wurde eine Standalone Eclipse RCP-Anwendung implementiert, die über die Funktionalität verfügt, einen Vergleichseditor mit zwei Profilen zu öffnen. Da bereits ein detailliertes Konzept besteht, dürfte die Implementierung dieser Komponente nicht allzu Zeitaufwendig sein. Vor allem, da Sie sich nicht in großem Maß von der Programmierung der Komponente für den Profilimport unterscheidet.

3.3.4 Filterprofileditoren

Standardmäßig werden in der DBWB die spezialisierten Editoren für Filterprofile anhand des Filtertyps gewählt. Ein AFP-Profil wird beispielsweise mit einem speziellen

[1] Ein Java Konfigurationsmechanismus

Editor für AFP-Profile geöffnet, der es dem Anwender ermöglicht, die Filterkonfiguration im Profil mit Hilfe einer grafischen Oberfläche vorzunehmen. In der Vergangenheit wurde der Dokumententyp des Profils anhand des Dateinamens ermittelt. Da mit dem Plug-in für den Profilimport dem Benutzer jedoch die Möglichkeit gegeben wird, importierte Profile in der Prozessperspektive umzubenennen(vgl. Abschnitt 3.3.2), musste eine Lösung gefunden werden, wie man Filterprofile anhand ihres Inhaltes einem Filtertyp zuordnen kann. Die Lösung fand sich in der Editierung der Plug-in-Manifest-Datei (plugin.xml). Statt den Dateinamen abzufragen wurde die Extension `org.eclipse.core.contenttype.contentTypes` überarbeitet und um einen Describer `org.eclipse.core.runtime.content.XMLRootElementContentDescriber` erweitert. Dieser erkennt das XML-Root-Element[1] und ordnet anhand des erkannten Typs einen Profileditor zu.

Listing 3.4: Bearbeitete ContentType-Extension am Beispiel des Formattyps AFP

```
<extension point="org.eclipse.core.contenttype.contentTypes">
      <content-type
            base-type="org.eclipse.core.runtime.xml"
            file-extensions="pro"
            id="net.compart.dbwb.filterconfigurator.mffprofile.afp"
            name="%contenttypes.mffprofile.afp.label"
            priority="high">
            <describer
                class="org.eclipse.core.runtime.content.
                XMLRootElementContentDescriber">
                      <parameter
                                name="element"
                                value="mffafp">
                      </parameter>
           </describer>
      </content-type>
 </extension>
```

3.3.5 Logging

Um den Ablauf von komplexen Programmen im Nachhinein verstehen zu können ist das Loggings von Programmzuständen ein wichtiger Teil der Softwareentwicklung. Im Softwareumfeld der DBWB existiert ein Logging Interface, das für alle Erweiterungen der Workbench genutzt wird. So wird das Verschmutzen der Software mit

[1] Äußerstes Element in einer XML-Datei

unterschiedlichen Logging-Konfigurationen vermieden. Es stehen zwei Arten von Loggern zur Verfügung. Der ContextLogger wird mit der zu loggenden Klasse initialisiert. Alle Meldungen werden sowohl in die, für den Benutzer vorgesehene, Log-Datei geschrieben. Der ContextLogger protokolliert auch in ein Trace-File. Der TechnicalLogger dagegen protokolliert dagegen ausschließlich in das Trace-File, das vor allem dem Support bei der Fehlerfindung helfen soll. Das Logging an sich unterscheidet sich in OSGi-Anwendungen nicht von dem in Standard Java Applikationen. Zwar bietet OSGi einen eigenen Log-Service, dieser verhält sich jedoch äquivalent und arbeitet ebenfalls mit verschiedenen Log-Leveln.

3.3.6 Verwendete Bibliotheken

In diesem Unterabschnitt sind die verwendeten Java-Bibliotheken aufgelistet und kurz beschrieben.

Bibliothek	**Funktion**
Java.io	Stellt Funktionen für Ein-und Ausgabestreams zur Verfügung. Auch Operationen auf dem Dateisystem und zur Serialisierung stellen einen wichtigen Teil dieser API dar.
java.util	Enthält das Collecion-Framework, Event Model, Zeit- und Datumsfunktionalitäten. Darüber hinaus bietet es viele Utility-Funktionen.
javax.xml.parsers	Stellt Parser zur Bearbeitung von XML-Dokumenten zur Verfügung. Folgende Parser werden Unterstützt: SAX (Simple API for XML), DOM (Document Object Model)
org.apache.commons.lang	Bietet statische Utility-Klassen, wie beispielsweise zur String-Modellierung.
org.w3c.dom	API des Document Object Models. Bietet Zugriff und Modifikationsmöglichkeiten auf Dokumente.
org.eclipse.osgi.util	Stellt Funktionalitäten bereit, um OSGi-Ressourcen handzuhaben.
org.eclipse.ui	API mit der das Eclipse RCP Benutzerinterface erweitert werden kann.
org.eclipse.swt	Stellt die Funktionalitäten des Standard Widget Toolkits bereit.
org.eclipse.jface	Ergänzt SWT um MVC (Model-View-Controller)-Ansatz
org.eclipse.core	Ermöglicht die Entwicklung von RCP-Komponenten im Zusammenspiel mit dem Dateisystem.

Tabelle 3.1: Verwendete externe Java-Bibliotheken

3.4 Ergebnisse

In diesem Abschnitt werden die Erkenntnisse und Ergebnisse dieser Arbeit dargestellt, zusammengefasst und bewertet.

3.4.1 Zusammenfassung

In dieser Arbeit stellte die Einarbeitung in verschiedene Technologien wie OSGi oder das RCP-Framework einen Schwerpunkt dar. Es wurde ein OSGi-Bundle im RCP-Umfeld entwickelt und implementiert, das den Anwendern der DocBridge® Workbench for Mill Plus die Möglichkeit verleiht, Filterprofile zu importierten und diese für die Konfiguration der In-und Outputfilter zu verwenden. Darüber hinaus wurde ein Konzept entwickelt, wie ein bestehendes Plug-in zur Konvertierung von Filterprofilen anhand eines gegebenen XML-Schemas, grafisch in die bestehende Oberfläche der DBWB eingebunden werden kann.

3.4.2 Bewertung

Die implementierte Softwarelösung entspricht im allgemeinen den Anforderungen, die an dieses Projekt gestellt wurden. Das Plug-in für den Import von Filterprofilen ermöglicht es dem Benutzer der DBWB, Profile von seinem Dateisystem zu importieren und durch das Ablegen der Dateien im ProfileRepository, diese für die Konfiguration der In-und Output-Filter zur Modifikation und Konvertierung von Dokumenten zu verwenden. Bei der Implementierung wurde darauf geachtet, dass die implementierte Lösung ein möglichst allgemeingültiges Verhalten aufweist. Das Plug-in beschränkt sich nicht auf einzelne Filtertypen, sondern deckt alle möglichen Formate ab. Auch die nicht funktionalen Anforderungen wurden erfüllt, da sich das Plug-in nahtlos in die bestehende Benutzeroberfläche einfügt. Somit behindert es den Nutzer nicht bei bereits bestehenden Arbeitsabläufen. Es erweitert die DBWB lediglich um eine weitere Funktionalität. Im Rahmen dieser Arbeit, bot OSGi ein ideales Werkzeug zur Erweiterung eines bestehenden, modularisierten Systems. Generell besteht bei der Anwendung einer Technologie Bedarf an Entwicklern mit Expertenwissen. Bei OSGi verhält es sich auch so. Die nötige Einarbeitungszeit wurde Anfangs unterschätzt, jedoch stellte sich die Unterteilung einer Anwendung in einzelne Bundles als sehr nutzbringend heraus und trägt zur Erweiterbarkeit der Applikation bei. Eclipse RCP

stellt ein mächtiges Frameworks bereit, mit dem sich spezialisierte Applikationen mit relativ wenig Aufwand entwickeln lassen.

3.5 Ausblick

Da sich in der Softwareentwicklung auch noch während der Implementierung die Projektanforderungen ändern können, spricht man oft davon, dass Software nie fertig wird. Auch bei dieser Arbeit stellte sich im Laufe der Entwicklung und Implementierung heraus, dass die Anforderungen im Nachhinein erweitert werden müssen. Zum jetzigen Zeitpunkt existiert keine Möglichkeit, default-Filterprofile aus dem ProfileRepository zu entfernen. Um das Profile Import Plug-in dem Endnutzer zur Verfügung zu stellen sollte jedoch eine solche Funktionalität implementiert werden, da sonst die Verwaltung der importierten Filterprofile zu unübersichtlich werden würde. Die Profile können momentan manuell vom Dateisystem und somit auch aus dem ProfileRepository gelöscht werden, jedoch entspricht dies nicht den Anforderungen an die Benutzbarkeit. Ein weiterer Punkt der in der Zukunft überarbeitet werden sollte ist das Umbenennen von Profilen beim Import in der Prozessperspektive. Da zusätzliche Profile des gleichen Typs aufsteigend nummeriert werden, kann dies zur Verwirrung des Anwenders führen. Besonders, wenn ein Profil mitten aus der Enumeration gelöscht und durch ein neueres ersetzt wird. Der Anwender soll in Zukunft seine Profile selbst benennen können. Aufgrund der Begrenzen Zeit des Projekts war es nicht möglich, Tests zu konzipieren und zu implementieren. Dies muss vor dem Praktischen Einsatz des Import-Plug-ins geschehen. Geplant sind GUI-Tests mit Jubula[1] und Cross-Plattform-Tests[2], da das Plug-in bisher ausschließlich auf Windows 64 Bit implementiert und getestet wurde.

[1] Eclipsebasiertes Tool zum automatisierten Testen von grafischen Benutzeroberflächen
[2] Tests auf verschiedenen Betriebssystemen z.B. Linux, Windows, Solaris

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

Daum, B. (2007), *Rich-Client-Entwicklung mit Eclipse 3.3. - Anwendungen, Plugins und Rich Clients*.

Ebert, R. (2004), *Eclipse RCP - Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client*. `http://www.ralfebert.de/archive/eclipse_rcp/EclipseRCP.pdf`.

Eclipse Magazin 3.10 (2010).

Kriens, P. (2008), How osgi changed my life, Technical report. `http://queue.acm.org/detail.cfm?id=1348594`.

OSGiAlliance (2012), Osgi service platform specification, Technical report. `http://www.osgi.org/Download/File?url=/download/r5/osgi.core-5.0.0.pdf`.

Popp, G. (2013), *Konfigurationsmanagement mit Subversion, Maven und Redmine: Grundlagen für Softwarearchitekten und Entwickler*, dpunkt.verlag GmbH; Auflage: 4.

Reichert, S. (2009), *Eclipse RCP im Unternehmenseinsatz: Verteilte Anwendungen entwerfen, entwickeln, testen und betreiben*, dpunkt Verlag; Auflage: 1.

Steve Northover, M. W. (2004), *SWT: The Standard Widget Toolkit, Volume 1*, Addison-Wesley Professional.

Abkürzungsverzeichnis

IDE	Integrierte Entwicklungsumgebung
RCP	Rich Client Platform
API	Application Programming Interface
OSGi	Open Services Gateway initiative
SWT	Standart Widget Toolkit
DBWB	DockBridge Workbench
GUI	Graphical User Interface
AFP	Apple Filing Protocol
PDF	Portable Document Format
XML	Extensible Markup Language
HTML	Hypertext Markup Language
JRE	Java Runtime Environment
ASE	Apache Software Foundation
JaxB	Java Architecture for XML Binding

Glossar

Framework

Programmiergerüst, das in der Softwaretechnik im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen verwendet wird.

Logging

(Automatische) Speicherung von Prozessen und Datenänderungen.

Mockup

Ein Mockup ist ein maßstäbliches Modell, das oft zu Präsentationszwecken oder zur Veranschaulichung genutzt wird.

Plug-in

Erweiterungsmodul, das von einer Software während ihrer Laufzeit erkannt und angeschlossen werden kann. Mit Plug-ins werden Applikationen um zusätzliche Funktionalitäten erweitert.

Repository

Verwaltetes Verzeichnis zur Speicherung und Beschreibung von digitalen Objekten (oft genutzt im Zusammenhang mit Versionsverwaltung).

String

Ein String ist eine Zeichenkette bzw. eine Folge von Zeichen aus einem definierten Zeichensatz.

Toolbar

Eine Toolbar (Statusleiste) ist eine waagerechte oder senkrechte Leiste mit kleinen, häufig bebilderten Schaltflächen, die als erweiternde Elemente der Menüführung von Programmen mit grafischer Benutzeroberfläche dienen.

Versionsverwaltung

System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird. Alle Versionen werden in einem Archiv oder Repository abgelegt, um ungewollte Datenverluste zu vermeiden.