Seminararbeit

Entwicklung einer generischen Embedded View Komponente auf Eclipse Basis für den IBM Workplace Client

Prof. Dr. Ludwig Nastansky

Sommersemester 2005

Betreuer:

Dipl.-Wirt. Inform. Ingo Erdmann

vorgelegt von:

Tobias Böker

Wirtschaftsinformatik

6114328

Ansgarstr. 36

33098 Paderborn

Abstract:
In today's PC work environment many different applications are necessary, in order to illustrate various kinds of information. For the purpose of creating a more flexible work environment, it is preferable to display this information in one overall context. Depending on the requirement, the type of information varies. Thus, it can be a matter of, e.g., pictures, videos, web pages or also *Notes Views*, which have to be embedded into the given context. The connection arises as a result of text, into which the information is embedded. Therefore, an environment which is able to process variable object types is necessary. The *IBM Workplace Client* is a new technology based on the *Rich Client Platform*. The basis for this is the freely available development environment *Eclipse*, which is arbitrarily expandable by its Plugin architecture. The *Workplace Client* already combines different applications under one user interface. The intention of this project is the development of a Plugin, that enables the user to embed the most different information objects in the given context.

Zusammenfassung:
Im heutigen PC-Arbeitsumfeld sind viele verschiedene Anwendungen nötig, um unterschiedliche Informationen darzustellen. Um eine flexiblere Arbeitsumgebung zu schaffen ist es wünschenswert, die Anzeige dieser Informationen in einem Kontext zu ermöglichen. Die Art der Information ist je nach Anforderung völlig unterschiedlich. So kann es sich dabei z. B. um Bilder, Videos, Webseiten oder auch *Notes Views* handeln, die in den gegebenen Kontext einzubetten sind. Der Zusammenhang ergibt sich durch Text, in den die Informationen eingebettet sind. Daher ist eine Umgebung notwendig, welche die unterschiedlichsten Objekttypen verarbeiten kann. *IBM Workplace Client* ist eine neue Technologie auf Basis der *Rich Client Platform*. Die Grundlage hierfür bildet die frei verfügbare Entwicklungsumgebung *Eclipse*, welche durch seine Plugin-Architektur beliebig erweiterbar ist. Der *Workplace Client* vereint schon jetzt verschiedene Anwendungen unter einer Oberfläche. Ziel dieser Arbeit ist die Entwicklung eines Plugins, dass dem Anwender die Einbettung unterschiedlichster Informationsobjekte in den gegebenen Kontext ermöglicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

OSGi	Open Service Gateway Inititative
RCP	Rich Client Platform
SWT	Standard Widget Toolkit
GUI	Graphical User Interface
WCT	IBM Workplace Client Technology
HTML	Hypertext Markup Language

1 Einleitung

1.1 Motivation und Zielsetzung

Um unterschiedliche Informationsobjekte darzustellen sind viele verschiedene Anwendungen notwendig. In einem Textdokument ist beispielsweise ein Link auf eine Webseite eingefügt worden, der ergänzende Informationen im Zusammenhang zum Text liefert. Soll diese Webseite jetzt angezeigt werden, muss dazu ein externes Programm, in diesem Fall der Webbrowser, aufgerufen werden. Dies führt dazu, dass die Webseite nicht direkt im gegebenen Kontext zur Verfügung steht. Groupwareplattformen wie *Lotus Notes / Domino* begegnen dieser Problematik mit *Embedded Views*, die es erlauben, eine Sicht (*View*) auf einen bestimmten Inhalt der zugrundeliegenden Datenbank in einen beliebigen Kontext einzubinden.

Ziel ist es, ein Konzept ähnlich der *Embedded Views* in *Lotus Notes* zu entwickeln. Dies soll es ermöglichen, Objekte in einen textuellen Kontext einzubetten. Diese Lösung soll generisch sein, damit jedes beliebige Objekt eingebettet werden kann. Das Konzept wird dann prototypisch als *Plugin* für den *IBM Workplace Client* umgesetzt. Der *IBM Workplace Client* ist das Frontend für die *IBM Workplace Technologie*. Dieser Client basiert auf der modular erweiterbaren *Eclipse* Plattform.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich in vier Haupt-Themengebiete. Im Anschluss an diese Einleitung wird ein Überblick über die verwendeten Technologien und Systeme gegeben und vermittelt das notwendige Grundwissen für die darauf folgenden Kapitel. In Kapitel 3 wird das Konzept der *Embedded View* und die Anforderungen an diese herausgearbeitet. Ausserdem wird gezeigt, welche Eigenschaften, sowohl inhaltlicher als auch technischer Art, erfüllt werden müssen, um eine generische Basislösung zu erstellen. Den Abschluss bildet Kapitel 4 mit der Umsetzung des Konzepts in ein prototypisches *Plugin*.

2 Grundlagen

2.1 Die Architektur der Eclipse Plattform

Eclipse besteht aus einer relativ kleinen Kernapplikation, die lediglich für die Ausführung der *Plugins* zuständig ist. Es werden praktisch alle Funktionalitäten werden als *Plugin* zum Kern hinzugefügt, was bedeutet, dass sich die Plattform beliebig anpassen lässt. Funktionalitäten können hinzugefügt bzw. entfernt werden. Zu einem *Plugin* gehört in jedem Fall die Manifest Datei. Diese wird als `plugin.xml` gespeichert, beinhaltet die Konfiguration des Plugins und beschreibt das Zusammenspiel mit der Plattform. Darüber hinaus ist die Anwendungslogik in Form einer `jar`-Datei sowie zusätzliche Dateien, wie z.B. Bilder oder Hilfetexte, enthalten.

2.2 Extension Points

Das Kernkonzept der *Plugin*-Architektur bilden die *Extension Points*, die im Manifest in beliebiger Anzahl definiert werden. Mit Hilfe der *Extensions* werden sie von anderen *Plugins* genutzt. Ein *Plugin* kann so die Funktionalität des anderen erweitern.[1] Die gesamte *Eclipse* Plattform basiert auf diesem Prinzip. Funktionen, wie beispielsweise ein Menüpunkt in der Benutzeroberfläche, werden über *Extensions* deklariert.

Die Verwaltung der *Extension Points* übernimmt die *ExtensionRegistry*. Sie ermöglicht dem Entwickler Zugriff auf die im System registrierten *Extension Points* und deren be-

[1] Vgl. [Gam03] S. 49 ff.

nutzenden Komponenten, also auf die *Plugins*, die den ensprechenden *Extension Points* verwenden. Ein wichtiger Punkt ist, dass die *Plugins* auch erst in den Speicher geladen werden, wenn ein Zugriff auf sie erfolgt. So kann eine *Eclipse* Umgebung beliebig viele *Plugins* enthalten, ohne dass unnötige Systemresourcen belegt werden.

2.3 Laufzeitkomponente

Eclipse verwendete früher eigene herstellerspezifische Formate für Laufzeitkomponenten und Plugins. Mit der Version 3 hat sich dies grundlegend geändert, denn intern wird jetzt der *OSGi*-Standard (*Open Service Gateway Inititative*) implementiert. Die *Open Services Gateway Initiative* ist ein im Jahre 1999 gegründetes Industriekonsortium. Sie legte den Standard von Diensten auf Komponentenbasis, die speziell in Netzwerken und eingebetteten Geräten zum Einsatz kommen soll, fest. OSGi Komponenten (*Bundles*) sind in der Eclipse Umgebung Java Programme, die bestimmte Dienste anbieten und über einen *OSGi*-Server verwaltet werden. In Eclipse übernimmt die Laufzeitumgebung die Rolle eines *OSGi*-Servers und die einzelnen Komponenten werden durch die Plugins repräsentiert. Die Laufzeitumgebung ist für alle Funktionen zur Bereitstellung und zu Verwaltung des Lebenszyklus der benötigten Klassen zuständig. Ein Dienst ist zum Beispiel die Menükomponente, die sich im Anwendungsfenster befindet. Eine Kompatibilätsschicht stellt sicher, dass auch ältere Plugins, welche nicht dem *OSGi*-Standard entsprechen, ausführbar bleiben.

2.4 Standard Widget Toolkit

Das *SWT*(*Standard Widget Toolkit*) stellt elementare GUI-Klassen zur Verfügung. Die Verwendung dieser Klassen erfolgt über plattformunabhängige Schnittstellen[2]. Die eigentliche Arbeit übernimmt eine betriebssystemspezifische und systemnahe Bibliothek, die sicherstellt, dass eine hohe Performance errreicht wird. Das *SWT* ist auf vielen Sy-

[2]Vgl. [Dau05], S. 147 ff.

stemplattformen verfügbar, wodurch der Entwickler sich nicht mit betriebssystemspezifischen Gegebenheiten auseinandersetzen muss. Ein wichtiges *Widget* ist das *Composite*. Es dient dem Gruppieren von beliebigen *Widgets* und fungiert somit als Container für beliebige Objekte, die der *SWT* Spezifikation entsprechen.

2.5 Die Eclipse Benutzeroberfläche

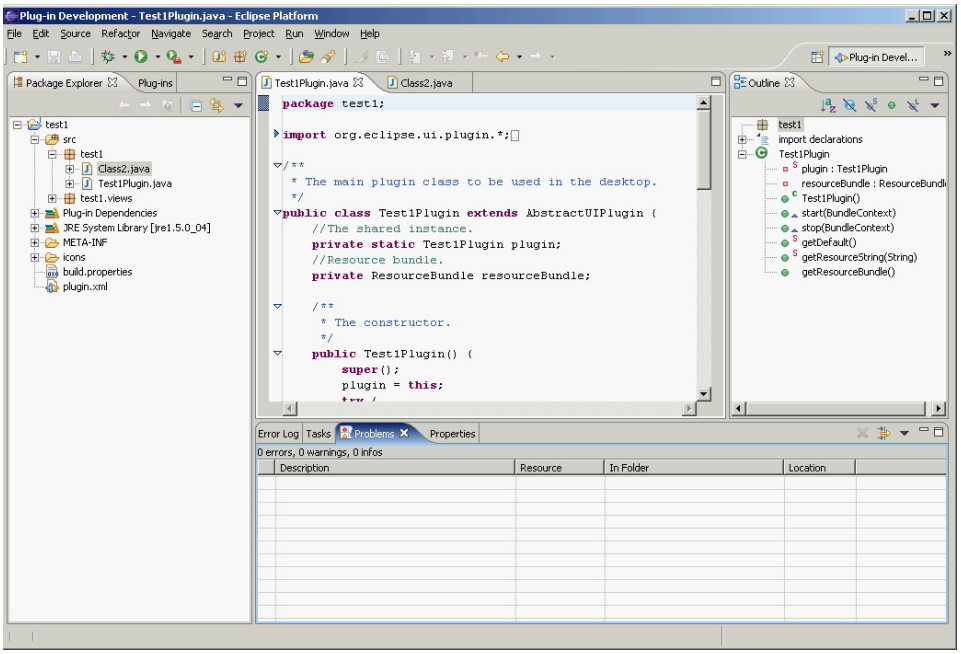


Abbildung 2.1: Die Eclipse Oberfläche

Das Hauptfenster von *Eclipse*, die sog. *Workbench*, setzt aus verschiedenen Elementen, wie der Menüleiste, Symbolleisten, einem Editorbereich und verschiedenen Ansichtsfenstern (*Views*) zusammen. Die Views können z.B. den Klassenaufbau des Projektes oder den Inhalt der Konsole darstellen. Eine Perspektive definiert den Aufbau der Views, Symbolleisten und Menüs in der Workbench.

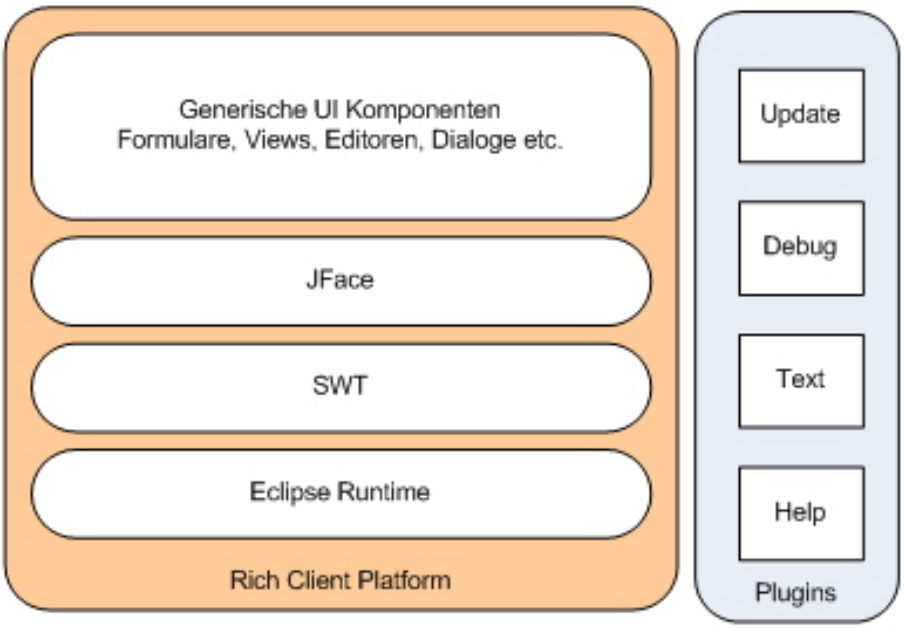


Abbildung 2.2: Rich Client Platform

2.6 Rich Client Platform

Unter einem *Rich Client* versteht man Software, die Anwendungslogik direkt im Client implementiert. Das Gegenstück dazu ist ein *Thin Client*, der keine anwendungsspezifische Logik enthält. Ein typisches Beispiel hierfür ist der Webbrowser, bei dem die Logik mit Hilfe von Webseiten dargestellt und auf dem Server ausgeführt wird.

Die *Rich Client Platform* (*RCP*) bildet, basierend auf der Eclipse Architektur und deren Standardkomponenten, zu denen auch die Laufzeitumgebung gehört, ein generisches Framework für die Entwicklung von Anwendungen. Die *RCP* fasst die Basisfunktionalitäten zusammen, die von allen Rich Clients benötigt werden. Sie beinhaltet die Eclipse-Runtime, das *SWT*, das JFace Framework, welches, basierend auf dem *SWT*, GUI-Komponenten zur Verfügung stellt und außerdem verschiedene UI-Elemente wie Formulare, Editor, Views, Dialoge etc.. [3]

[3] Vgl. [Dau05], S. 551-560

2.7 IBM Workplace Client Technology

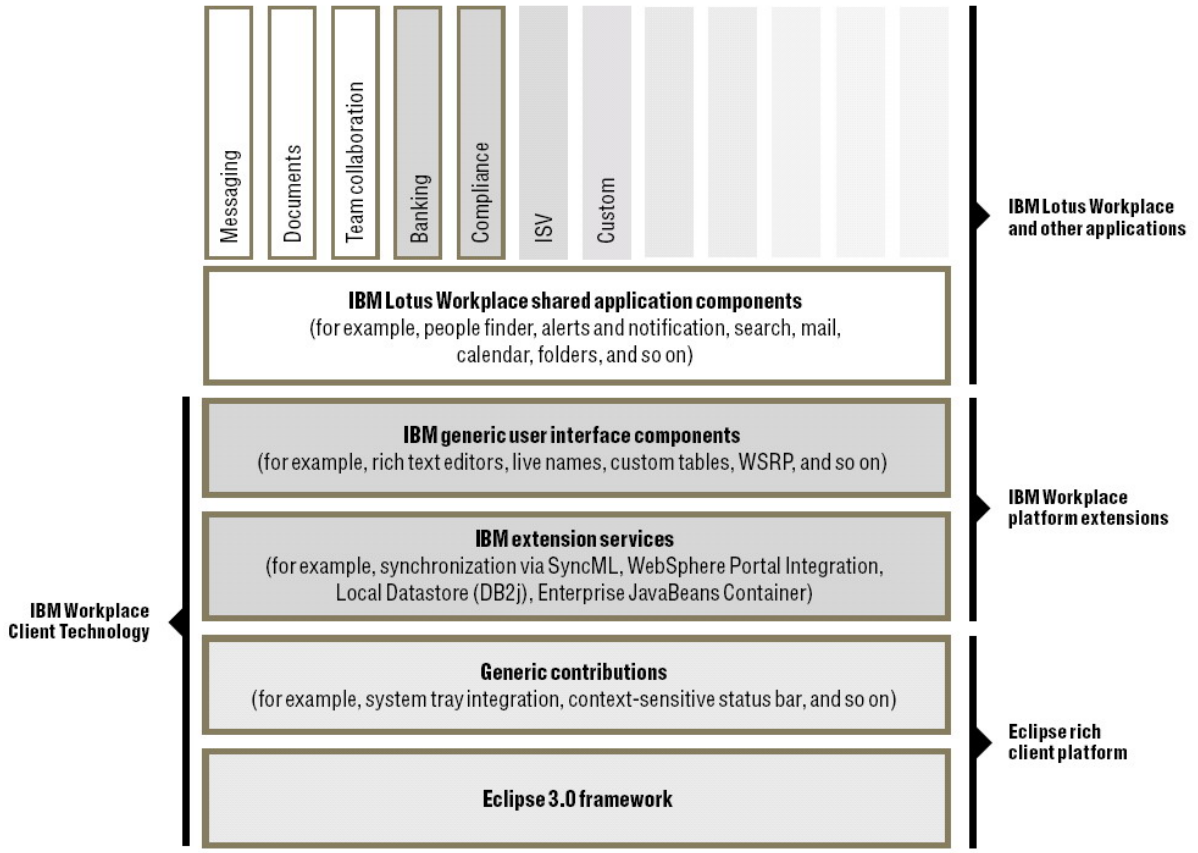


Abbildung 2.3: IBM Workplace Architektur

Die *IBM Workplace Client Technology*[4] (*WCT*) erweitert die *RCP* mit einigen Plugins. Diese umfassen Serviceprogramme, wie Synchronisation oder lokale Datenbanken, und Komponenten für die Benutzeroberfläche, wie Rich Text Editoren. Durch diese Erweiterungen entsteht eine integrierte Arbeitsumgebung, die Funktionalitäten rund um den *IBM Workplace* bereitstellt.

[4]Vgl. S. 9 [IBM04]

2.8 Embedded Object

EmbeddedObjects sind Objekte, die in einen Text eingebettet werden können. Die Art der Information ist je nach Anforderung völlig unterschiedlich. So kann es sich dabei z. B. um Bilder, Videos, Webseiten oder auch *Notes Views* handeln. Die *Notes Views* können als *Embedded View* in Formularen oder Seiten eingebettet werden.

3 Konzept

3.1 Embedded View Komponente

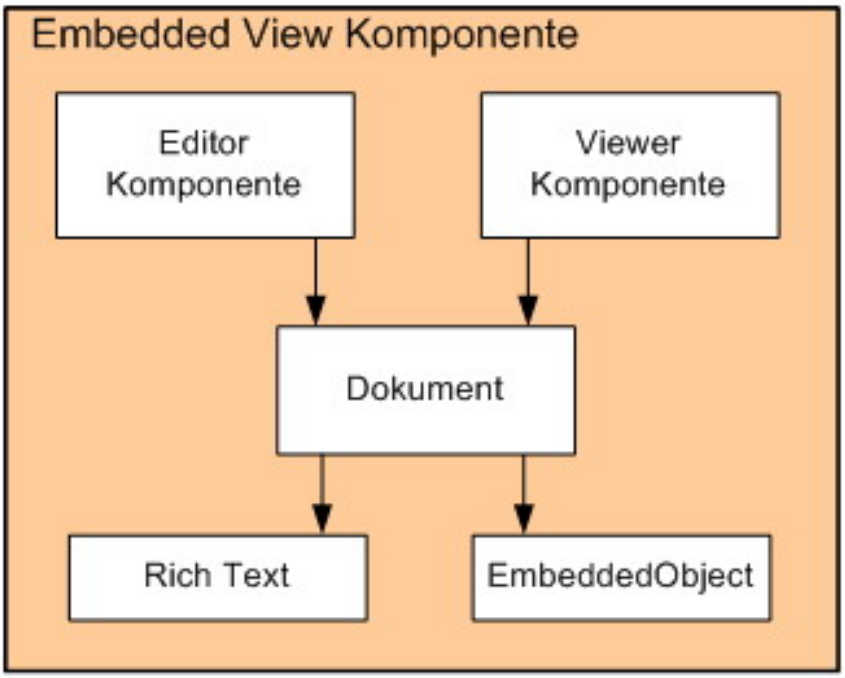


Abbildung 3.1: Embedded View Komponente

Das Konzept lehnt sich an das *Compound Document* an. Diese Documente dienen als Container für strukturierte und unstruktierte Daten [Fis02]. [1] Abbildung 3.1 zeigt den

[1] Vgl. Fischer et al. 2002, S. 252 ff.

Aufbau der gesamten Embedded-View Komponente, die im Wesentlichen aus drei Einzelkomponenten besteht. Zentraler Punkt ist das Dokument, dass die Informationen beinhaltet. Dieses Dokument wird mit Hilfe der Editorkomponente erstellt. Das Dokument ist ein Container für Informationen und enthält den statischen Text als Richtext sowie die einzubettenden Objekte, genauer gesagt die Meta-Daten zu diesen Objekten. Die Anzeige des Dokumenteninhalts und der eingebetteten Objekte übernimmt die Viewer Komponente. Sie zeigt den Text an und erzeugt aus den Meta-Daten des eingebetteten Objekts die jeweiligen konkreten Objekte.

3.1.1 Eigenschaften und Anforderungen

Die *Embedded View* Komponente als Ganzes muss verschiedene Anforderungen erfüllen. Sie benötigt die *Editor Komponente* um den statischen Inhalt des Dokuments zu erstellen. Um lediglich das reine Konzept umzusetzen, genügt es, dem Anwender flachen Text für den statischen Inhalt anzubieten. Dennoch werden hier einige einfache Formatierungsmöglichkeiten benötigt. Diese Dokumente können erstellt, gespeichert und geladen werden. Der Editor sollte die Möglichkeit bieten, ein *Emdedded Object* in den Text einzufügen. Diese Objekte sollen vom Anwender bearbeitet werden können, was die Implementierung einer entsprechenden Funktionalität bedingt.

3.1.2 Konfiguration

Der Viewer ist zuständig für die Erstellung and Anzeige der *Embedded Objects*. Zur Erfüllung seiner Aufgabe benötigt er einige Informationen zum Objekt. Der Hauptparameter ist der Typ des Objektes. Einige Objekttypen benötigen noch zusätzliche Informationen bzw. Parameter, die für die korrekte Ausführung der Objektfunktionalitäten notwendig sind. Das Objekt muss in der Lage sein, Art und Anzahl der Parameter zur Laufzeit mitzuteilen. Sämtliche Informationen zum Objekt werden direkt im Dokument abgelegt. So kann der Viewer aus dem Dokument und dem Inhalt alle notwendigen Objekte erstellen.

3.2 Generische Aspekte

Generische Programmierung im Allgemeinen ist eine Art der Programmierung, bei der Funktionen und Klassen möglichst allgemein verfasst werden. So können sämtliche Methoden auf unterschiedliche konkrete Typen angewandt werden. Die Methoden sind dabei nicht für einen bestimmten Typen geschrieben, sondern stellen lediglich Strukturanforderungen an den zu verarbeitenden Typ. Somit ist eine hohe Wiedervewendbarkeit gegeben, da vorhandene Methoden und Funktionen mit allen Datentypen funktionieren. Der generische Aspekt in Zusammenhang dieser Arbeit bezieht sich auf die Wiederverwendbarkeit der einzubettenden Objekte. Anforderungen an diese Objekte müssen so allgemein wie möglich beschrieben werden.

3.3 Schnittstellen

3.3.1 Interfaces für Standardisierung

Schnittstellen dienen dem Austausch von Informationen zwischen den einzelnen Komponenten. Sie werden durch Schnittstellenbeschreibungen definiert, welche die Anforderungen an die Schnittstellen enthalten. Unterstützen Elemente die gleiche Schnittstelle, so können diese gegeneinander ausgetauscht werden. Damit ein Objekt eingebettet werden kann, müssen gewisse Anforderungen erfüllt sein. Da einige Objekte für ihre korrekte Funktion Parameter benötigen, muss eine Schnittstelle zur Parameterübergabe festgelegt werden. Außerdem muss das Objekt eine Möglichkeit haben, dem Container mitzuteilen, welche Parameter es benötigt.

3.3.2 Kommunikation mit einem eingebetteten Objekt

Die Kommunikation des Containers mit dem einzubettenden Objekt verläuft unidirektional.

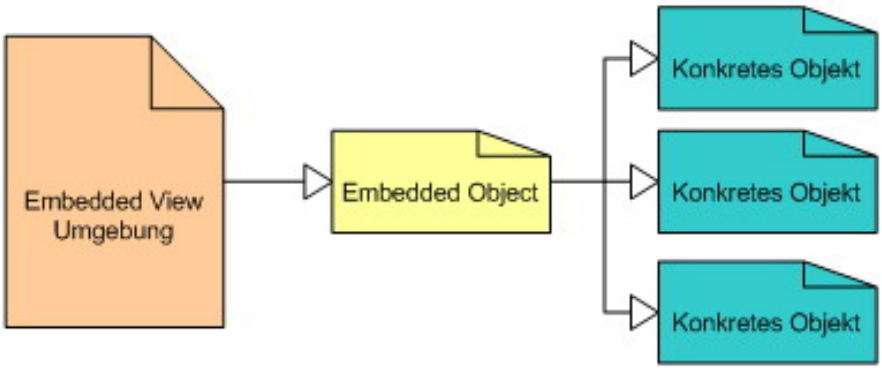


Abbildung 3.2: Kommunikation mit dem Embedded Object

Der Container erzeugt ein neues Objekt und teilt ihm die Umgebung mit, in der es existiert, bzw. in der es seine grafische Ausgabe durchführt. Das Objekt bekommt die Parameterliste übergeben und führt alle Operationen ohne Zugriff auf den Container aus. Damit ist das eingebettete Objekt als autonome Einheit zu sehen.

4 Implementierung

4.1 Benutzeroberfläche

Die Implementierung des *Embedded View* Plugins erfolgt in der Programmiersprache Java. Als Ablaufframework wurde die Eclipse Rich Client Platform verwendet, die dem Entwickler besonders für die Benutzeroberfläche viele grundlegende Funktionen zur Verfügung.

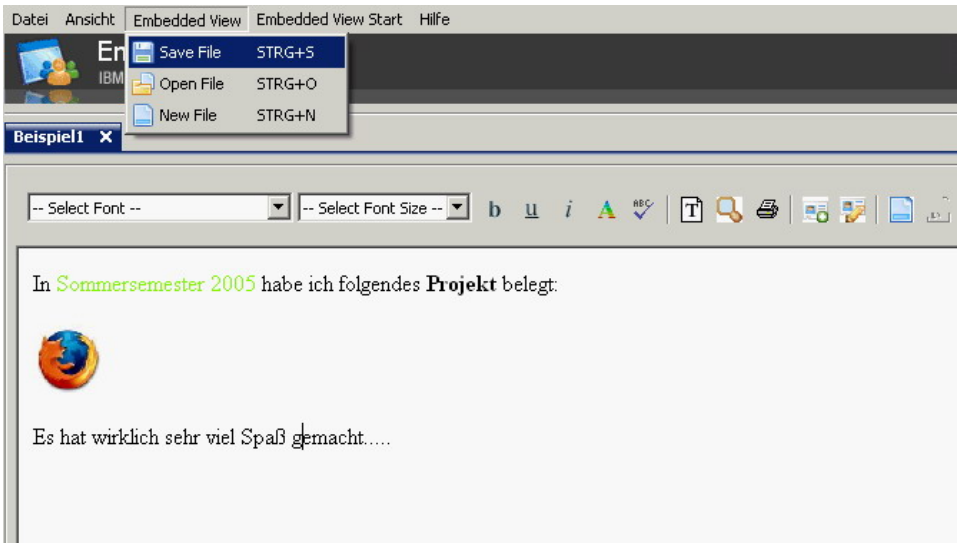


Abbildung 4.1: Die Editor Komponente

Abbildung 4.1 zeigt das Menü der Anwendung und die Editor Komponente. Das Menü bietet mit *New File*, *Open File*, *Save File* die Grundfunktionen der Dateiverwaltung. Die Editor Komponente besteht aus einer Menüleiste und einem Feld zur Texteingabe. Die Menüleiste enthält Funktionen zur Textformatierung, zum Erstellen und Bearbeiten

der eingebetteten Objekte sowie eine Schaltfläche zum Starten der Viewer Komponente, die in Abbildung 4.2 dargestellt ist.

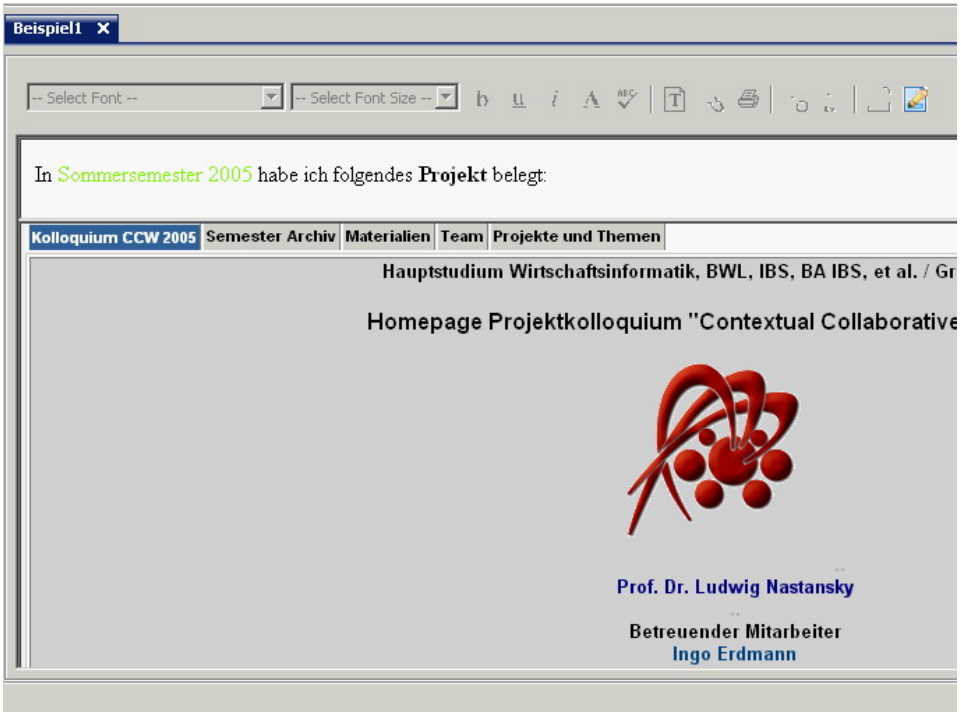


Abbildung 4.2: Die Viewer Komponente

Diese Abbildung zeigt die Viewer Komponente. Die Menüleiste enthält jetzt nur noch die Funktion zum Aufrufen der Editor Komponente. Der Platzhalter für das einzubettende Objekt wird hier durch eine Browser Komponente ersetzt, die eine Webseite anzeigt.

Wählt der Benutzer die Funktion `Insert Embedded Object` aus der Menüleiste erscheint der in Abbildung 4.3 gezeigte Dialog. Alle im System registrierten Objekte werden in einem Kombinationsfeld angezeigt. Bei erfolgter Auswahl muss der Anwender, je nach Objekttyp, die Parameterwerte in der Tabelle vervollständigen.

Die Funktion `Edit Embedded Object` funktioniert, bis auf die Tastsache, dass nicht alle um System registrierten, sondern im Dokument verwendeten Objekte aufgelistet werden. Auch hier können die Parameter verändert werden, das Objekt wird daraufhin im Dokument aktualisiert.

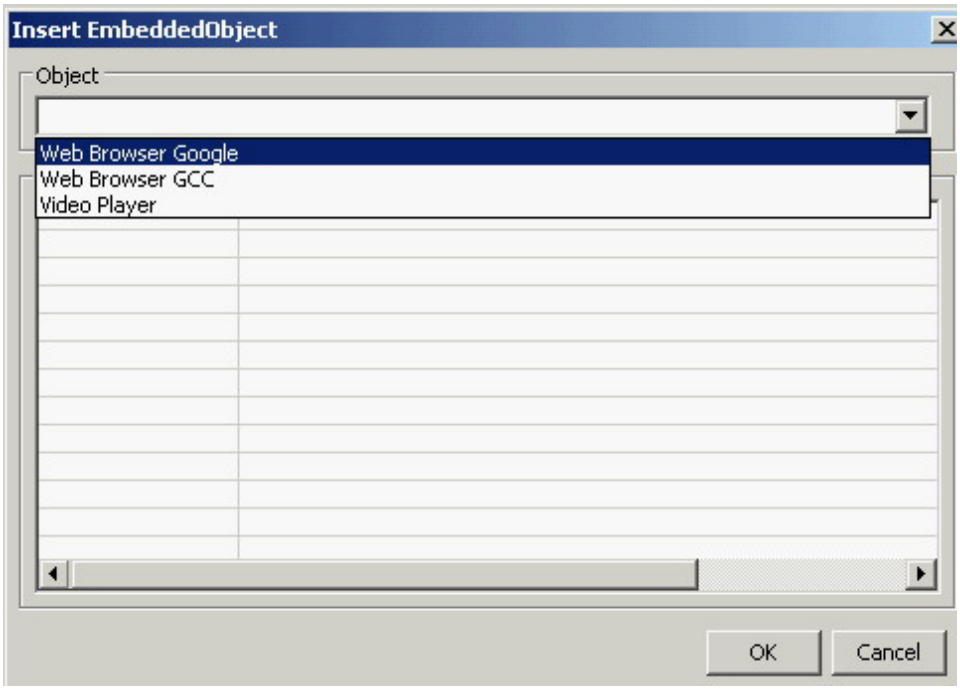


Abbildung 4.3: Embedded Objekt einfügen

4.2 Schnittstellen

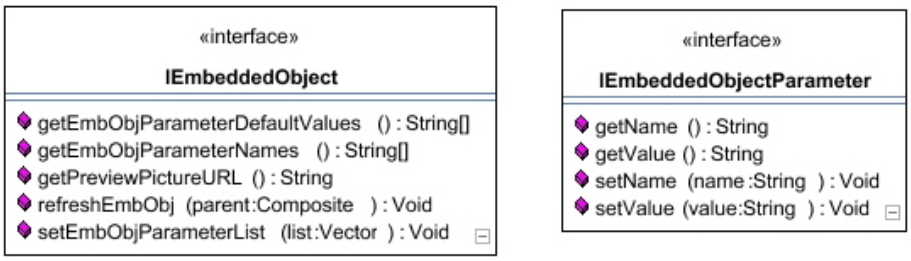


Abbildung 4.4: Schnittstellen

Zur Standardisierung werden *Java Interfaces*[1] verwendet. Ein *Embedded Object* muss `IEmbeddedObject` erfüllen. In diesem *Interface* werden einige Methoden festgelegt, die nachstehend kurz erläutern werden.

[1] Vgl. [Bar03] S. 344 ff.

- `getEmbObjParameterDefaultValues`: Gibt ein String Array mit den Standardwerten der Parameter zurück.
- `getEmbObjParameterNames`: Gibt ein *String Array* mit den Namen der Parameter zurück. Diese werden vom *Einfügen*-Dialog für die Auflistung verwendet.
- `getPreviewPictureURL`: Gibt ein *String* mit dem Pfad zum Vorschaubild für den Editor zurück.
- `refreshEmbObj`: Diese Methode wird aufgerufen, wenn das Objekt angezeigt wird. Übergeben wird ein *SWT Widget Composite*. Dieses *Composite* dient als Container für die Grafikausgabe des *EmbeddedObject*. Hier findet auch die Parameterauswertung statt.
- `setEmbObjParameterList`: Übergibt einen *Vector* mit der Parameterliste. Die einzelnen Elemente des *Vectors* erfüllen das *Interface* `IEmbeddedObjectParameter`.

Das *Interface* `IEmbeddedObjectParameter` dient der Übergabe der Parameterliste. Jedes Element dieser Liste erfüllt das *Interface* und besitzt eine `getName` und eine `getValue` Methode. Durch die Übergabe der Parameter als Liste und nicht als *String Array* ist eine sichere Verarbeitung durch das *EmbeddedObject* gewährleistet, da der Name und der Wert des jeweiligen Parameter abgefragt werden können. Das hat zur Folge, dass die Reihenfolge der Parameter nicht von Bedeutung und die Verarbeitung weniger fehleranfällig ist.

4.3 Klassenstruktur

Im folgenden werden die wichtigsten Klassen des Systems grafisch dargestellt (vgl. Abbildung 4.5) und genauer beschrieben.

- `EmbeddedView` ist die eigentliche Hauptklasse der *Embedded View Komponente* und erweitert *ViewPart*, die Basisklasse der *Eclipse View*.

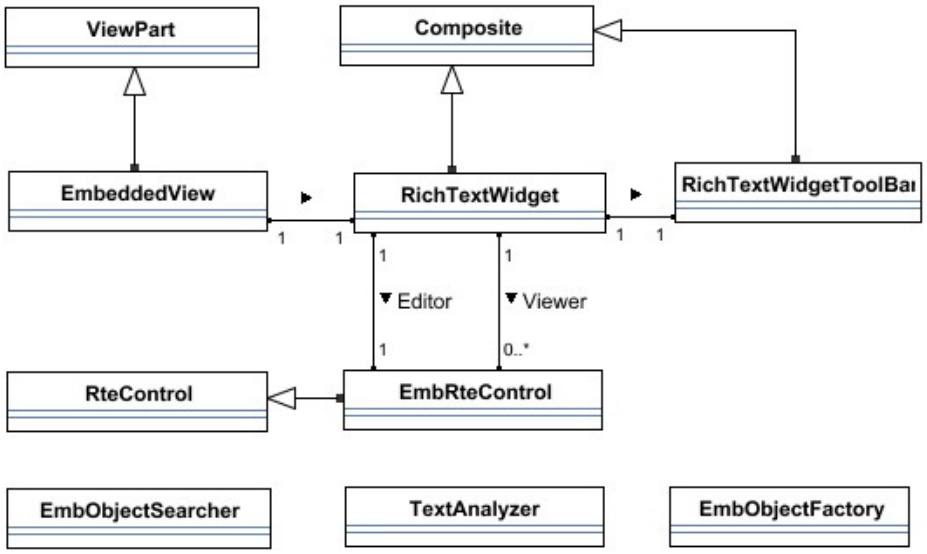


Abbildung 4.5: Klassendiagramm

- `EmbRteControl` erweitert die Rich Text Editor Klasse *RteControl*, welche zum Umfang des IBM Workplace Client gehört, mit notwendigen Methoden für die *Embedded View Komponente*.
- `RichTextWidget` erweitert die *SWT* Klasse *Composite*. Diese Klasse steht in direktem Zusammenhang mit der *EmbeddedView* Klasse. Sie ist sowohl Editor- als auch Viewer Komponente. Im *Editor Modus* wird genau eine Instanz des *EmbRteControl* zur Textverarbeitung genutzt. Im *Viewer Modus* ist die Zahl der Instanzen gleich der Anzahl der Textblöcke des gesamten Textes.
- `RichTextWidgetToolbar` erweitert die *SWT* Klasse *Composite*. Diese Klasse beinhaltet die Menüleiste der *Embedded View Komponente*.
- `EmbObjectSearcher` ist für den Zugriff auf die *ExtensionRegistry* zuständig und liest alle zur Laufzeit verfügbaren *EmbeddedObject* Extensions aus.
- `TextAnalyzer` analysiert den Text des Dokuments und teilt diesen in *Embedded-Object*- und Textregionen auf.
- `EmbObjectFactory` dient als Hilfsklasse zum Erzeugen des konkreten *Embedded-Object*.

4.4 Anmerkungen

Das *EmbRteControl* verwaltet den Inhalt intern als *HTML Code*. Dazu musste eine Lösung geschaffen werden, das *EmbeddedObject* im Text als solches zu markieren. Realisiert wird das mit dem *HTML* Grafik Tag *<img>* und dem Attribut *longdesc*. In dieses Attribut wird die Textkennzeichnung für das *EmbeddedObject* geschrieben. Diese Technik bietet zusätzlich noch den Vorteil, dass im Editor das Bild und nicht die Kennzeichnung erscheint.

Die *Eclipse* Umgebung stellt dem Entwickler Standard Editoren zur Verfügung. Diese können aber im *WCT* nicht verwendet werden, da hier keine Editorbereiche in Perspektiven unterstützt werden und somit die Texteingabe mit Hilfe des *RteControl* erfolgt.

Die *Viewer Komponente* zerlegt den Inhalt des *EmbRteControl* mit Hilfe der *TextAnalyzerKlasse*. Für Textregionen wird jeweils eine neue *EmbRteControl* Klasse erzeugt. Für Objektregionen wird ein *SWT Composite* erzeugt und dem Objekt als Parameter in der Methode `refreshEmbObj` übergeben. Dieses *Composite* kann dann zur Grafikausgabe verwendet werden. Das Layout der *Preview Komponente* zeigt Textregionen immer über die komplette Seitenbreite und über die notwendige Höhe an. Objektregionen können in der Breite und Höhe variiert werden. Das Objekt kann dieses steuern, indem es die Größe des *Composites* verändert.

Die Verwendung des *Extension Point* `de.gcc.embiew.EmbeddedObject` der *Embedded-View Komponente* erfordert, dass die verwendete Klasse das *Interface* `IEmbeddedObject` erfüllt. Dadurch wird gewährleistet, dass keine unzulässigen Objekte erzeugt werden.

5 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein Konzept entwickelt, das es ermöglicht, verschiedene Arten von dynamischen Objekten in eine statische Umgebung zu integrieren. Die Funktionalität der entwickelten Komponente reicht von Richt Text Bearbeitung und einfachen Dateiverwaltungsfunktionen bis hin zum Einfügen und Bearbeiten von eingebetteten Objekten. Die Verwendete *Rich Client Platform* bietet eine vielversprechende Grundlage im Hinblick auf Weiterentwicklungen für den *IBM Workplace Client*.

Sinnvolle Erweiterungen könnten dazu beitragen, die Komponente komfortabler und Benutzerfreundlicher zu gestalten.

Zwei zentrale Punkte sind ein flexibleres Layout in der Darstellung der *EmbeddedObjects* sowie die Integration in die Dokumentenverwaltung des *IBM Workplace Client*. Dadurch ist es möglich, die Dokumente zentral Serverseitig zu verwalten und somit vielen Benutzern gleichzeitig zur Verfügung zu stellen.

6 Literaturverzeichnis

[Bar03] Kölling Barnes. *Objektorientierte Programmierung mit Java*. Pearson Studium, 2003.

[Dau05] Daum. *Java - Entwicklung mit Eclipse 3 - Anwendungen, Plugins und Rich Clients*, volume 2. dpunkt Verlag, August 2005.

[Fis02] Dangelmaier Nastansky Suhl Fischer, Herold. *Bausteine der Wirtschaftsinformatik, 3. Auflage*. Erich Schmidt Verlag, 2002.

[Gam03] Beck Gamma. *Contributing to Eclipse*. Addison Wesley, 2003.

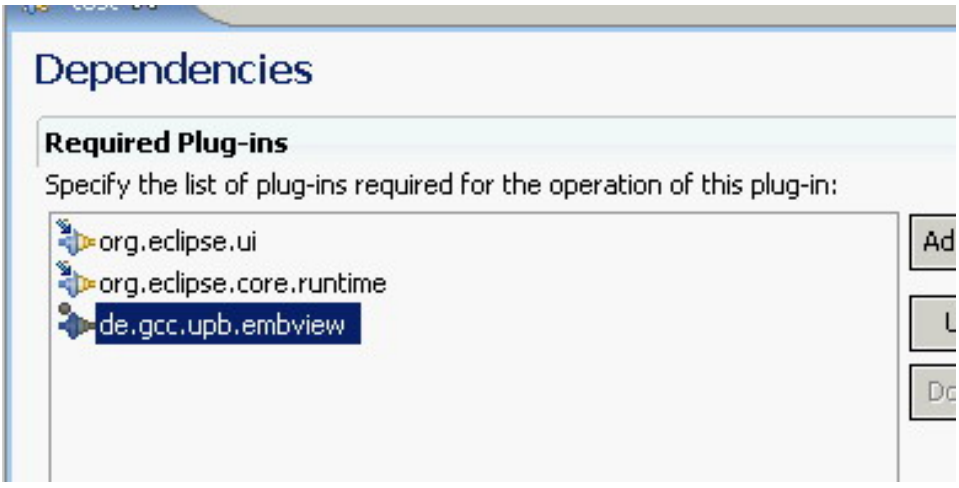
[IBM04] IBM. Ibm workplace client technology strategy white paper. 2004.

7 Anhang

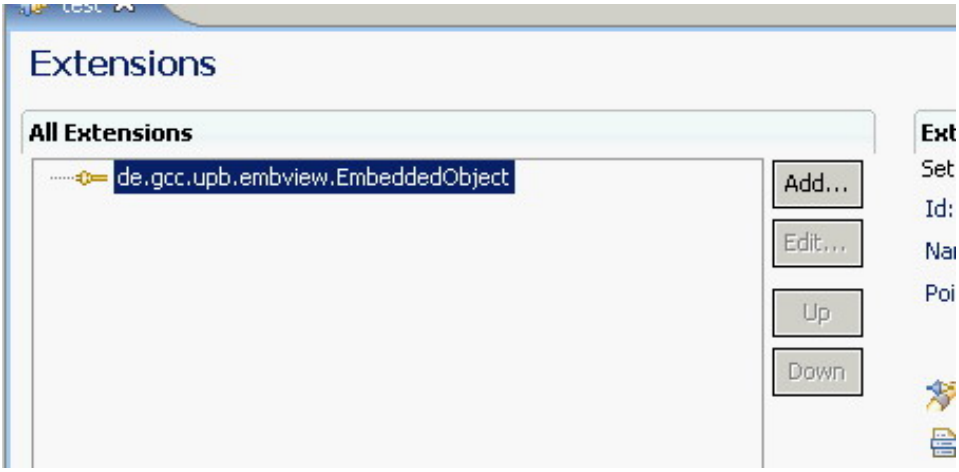
7.1 Exemplarische Vorgehensweise zum Erstellen eines EmbeddedObjects

Hier wird kurz exemplarisch beschrieben, wie ein *EmbeddedObject* erstellt werden kann.

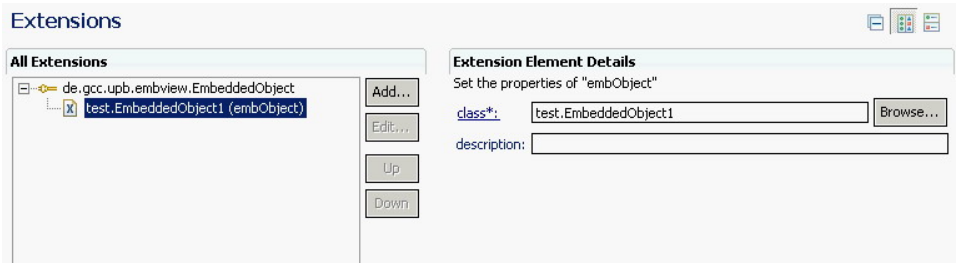
1. Neues Plugin-Projekt erstellen oder bestehen Projekt öffnen.
2. Das Plugin `de.gcc.upb.embview` im Manifest in die *Dependencies* aufnehmen.



3. Extension für den Extension Point `de.gcc.embiew.EmbeddedObject` erstellen.



4. `embObject` Extension einfügen



Beschreibung im Feld *description* eingeben. Diese wird später im EmbeddedView Dokument angezeigt. Klasse erstellen: Existiert schon eine Klasse für dieses Objekt, so kann sie mit `Browse...` ausgewählt werden. Sonst den link *class** auswählen und den Klassennamen angeben. Die Klasse für das Objekt wird dann direkt mit dem zu erfüllenden *Interface* angelegt.