Analyse des Model-View-Controller Paradigmas zur Unterstützung des Multimedia Retrievals in Mobiltelefon

Studienarbeit

Ke Huang

Hamburg, Juni 2008

Themensteller: Prof. Dr. Ralf Möller

Betreuerin: M.Sc. Irma Sofia Espinosa Peraldi

Institute für Softwaresysteme (STS)

Technische Universität Hamburg-Harburg (TUHH)

1 Motivation und Einleitung

1.1 Multimedia Retrieval im Mobiltelefon

Mobiltelefon ist derzeit nicht nur als ein Kommunikationsmittel sondern bereits als ein mobiles Multimediagerät im alltäglichen Leben benutzt. Leute können mit dem Mobiltelefon Internet surfen photographieren, Video und Musik spielen usw. Die Herstellers des Mobiltelefons möchten alle Multimediatechniken im Mobiltelefon integrieren, damit können die Benutzern irgendwo und irgendwann das digitale Technik genießen.

Aber außer den im Mobiltelefon gespeichert Multimedia-Daten möchte man häufig auch die Information über die Inhalte der Multimedia-Daten weiter von dem Internet wissen. Kann man ein inhaltsbasiertes Multimedia-Suche im Mobiltelefon machen? damit nicht nur die Benutzern sondern auch die Informationen durch das Mobiltelefon mit äußerlicher Welt kommunizieren können. Mit dem semantischen Webs wird es ermöglicht. Jede Multimediaressource wird in dem semantischen Web durch eine formale und maschinenlesbare Metadatei über seine Inhalte beschrieben und ist durch die Web-Geräte und Query abfragbar. Aber wie wird solche Anwendung des semantischen Webs im Mobiltelefon umgesetzt? Dazu wird das MVC-Paradigma als Entwurfsmuster für Entwurf des Programms im Mobiltelefon verwendet.

1.2 MVC-Paradigma im Mobiltelefon

Das Konzept des MVC-Paradigmas ist gemäß Aufgabeverteilung zur Zerlegung der Programmcode in drei voneinander unabhängige Einheiten ***Model***, ***View*** und ***Controller***.

- Das Model-Objekt fasst die Anwendungsdaten und Anwendungsfunktionen zusammen.
- Das View-Objekt stellt eine Bildrepräsentation vom Modelobjekt in Ausgangeinrichtung dar.
- Das Controller-Objekt bietet jede Benutzeraktion eine Funktionsschnittstelle, die durch Verwalt der Methoden von Model-Objekte und View-Objekte die entsprechende Aufgabe implementiert.

Das Ziel von solcher Architektur ist die Benutzerdaten und Datenrepräsentation zu entkoppeln. Damit ist Austauchen der Datenrepräsentation und Wiederverwendung der Benutzerdaten ermöglich, Änderung und Erweiterung des Programms wird leichter. Die Funktionalitäten von dem gleichen Model-Objekt kann ohne Änderung durch verschiedenen Repräsentation dargestellt. Die Programmcode können wiederholt benutzt werden, daraus spart viel Zeit und Kosten in der Softwareentwicklung.

Heute wird das MVC-Paradigma bereits als ein wichtiges Entwurfsmuster in vielen GUI-basierten Anwendungen und Webdienste durch verschiedene Programmiersprachen sehr umfassend verwendet. Aber sind MVC-Paradigma meistens nur im PC für objektoritierte Programmentwicklung angewendet, zum Beispiel in Java wird MVC-Paradigma zahlreich in J2EE und J2SE für große bzw. normale Softwarenentwicklungen angewendet, aber ist deren Anwendung sehr selten in J2ME für Mobiltelefons gesehen.

Der Grund liegt darin, dass wegen Beschränkung der Hardware die Applikationen des Mobiltelefons relativ klein und einfach sind. Die Verwendung des MVC-Paradigmas wird zu einem große Code-Volumen und einer komplexe Programmstruktur führen. Aber mit schneller Entwicklung der eingebetteten Hardware ist derzeit die Geschwindigkeit und Kapazität der Hardware immer schneller bzw. größer geworden. Das Mobiltelefon kann nicht nur die einfache Applikationen laufen, deren Code innerhalb 64 KB beschränkt, sonder auch die größere Multimedia-Applikation, mehr als 1MB, ausführen.

Die meiste Teil von einem kleinen J2ME Applikation ist die Bilddaten und Audiodaten, Programmcode besitzt nur einen kleinen Teil. Um die Leistung zu erhöhen wird häufig ganze Programmcode nur in einer Klassen geschrieben, deren Rückrufmethode *commandAction(Command c, Displayable d)* implementiert alle Zustand- und Anzeigewechseln. Mit solcher Struktur kann man kleinstes Jar-Packet und schnellste Reaktionsgeschwindigkeit bekommen. Aber wenn die J2ME-Applikation viele graphische Benutzeroberflächen und Datenverkehr hat, hat diese schlanke Struktur viele Nachteile. Einerseits, Konstruierung der tausend Codezeilen in einer Klass ist für Änderung und Erweiterung des Programms nicht einfach und leicht. Andererseits, Alle Benutzeroberflächen sind aufeinander in einer Klasse gekoppelt, es führt zu größte Schwierigkeit zu Austausch und Wiederverwendung einer Benutzeroberfläche.

Dieser Mangel ist genau von MVC-Paradigma behebbar. In dieser Studienarbeit wird mit dem Thema auseinandergesetzt, wie das MVC-Paradigma zur Implementierung eines J2ME-Programms für Multimedia-Retrievals verwendet wird.

2 Multimedia Retrieval im Mobiltelefon

2.1 Information Retrieval in unserer Zeit

Mit der Explosion der digitalen Multimedia Information und Netzwerktechnik werden viele Multimedia Ressourcen in digitale Form umgewandelt. Damit sind das Suchen und die Übertragung der Multimediaresourcen durch Internet leichter geworden. Durch Eingabe von Stichwörter über Informationsinhalte in einer Suchmaschine wie Google und Yahoo kann man sehr einfach die von den Stichwörtern abhängigen Ergebnisse finden. Aber ist häufig der Ergebnisseraum sehr groß und die meiste Inhalte sind nicht genau von dem Benutzer gefordert. Da es viele Inhaltdarstellungen und Relationen zwischen den Stichwörter gibt, die von Suchmaschinen nicht verstanden werden. Die Multimedia-Ressourcen werden zurzeit nur mit Stichwörtern als Metadaten in der Internet identifiziert. Die Suchmaschinen können nur mit solchen Metadaten die Webresource im Internet lokalisieren.
Außerdem die Kombination der Stichwörter beeinflusst auch das Suchergebnis. Sogar wenn man die richtige Stichwörter und Kombination wählt, verkleinert sich nur die Suchergebnisraum. Man kann auch nicht dem exakten Ergebnis bekommen. Kann der Informationsinhalt von Maschinen verständlich beschrieben und exakt im Internet abgefragt werden? Da zu wird das semantische Web von Tim Berners-Lee, dem Erfinder des Word Wide Web, im Jahre 1998 vorgeschlagen.

2.2 Semantisches Web

Das semantische Web ist ein Erforschungsthema von W3C, welche als eine Erweiterung von World Wide Web in der Zukunft betrachtet und ermöglicht den Internetbenutzern über die Grenze der verschiedenen Informationsträger wie Applikation und Webseite die Daten zugreifen und gemeinsam benutzen zu können.

In dem semantischen Web werden nicht nur die Inhaltelementen der Daten sondern auch dazwischen liegende Beziehungen in den maschinenlesbaren Daten formal geschrieben. Damit können die Inhalte der Informationsressourcen besser von den Maschinen identifiziert und bearbeiten. Weitere Vorstellung über semantisches Web findet man in der Webseite von W3C***[IoSW].***

2.3 Ontologie und Beschreibungslogik

Diese auf den Inhaltelemente und Beziehungen basierte formale Informationsdarstellung nennt man „***Ontologie***“, welche durch Beschreibungslogik mit Abox und Tbox für den abstrakte Begriffe und konkrete Objekte der realen Welt definiert.

Tbox *ist eine Menge Axiome zur Beschreibung von Begriffen (Konzepten) und ihre Beziehungen.*
Abox *ist eine Axiom zur Beschreibung von Objekten(individuaen) und ihre Beziehungen.*

Unter Ontologie wird jede Media-Ressource durch Abox und Tbox über die Attributen seiner Inhaltelemente exakt beschrieben und in der Web ausschließlich kennzeichnet. Dadurch die umfangreiche ähnliche Suchergebnisse vermeidet werden können.

Im Computerwissenschaft und künstliche Intelligenz werden Ressource Deskription Framework(RDF) und Web Ontology Language(OWL) als formale Sprache für Aufbau der Ontologie verwendet. Die mit RDF und OWL zur Beschreibung der Informationsressourcen verwendete Daten heißt „***Ontologie-Metadaten***“. Das Information Retrieval ist durch solche Metadaten die Informationsressourcen lokalisieren. Mit Anwendung der Ontologie-Query kann man die von Ontologie-Metadaten beschriebene Informationsressourcen exakt abfragen.

2.4 Bildbasierte Multimedia Retrieval

Das Ziel von dem semantischen Web ist es intelligente Verwaltung der Webdaten durch Maschinen zu ermöglichen. Damit die Informationssucht und Informationsnutzung mehr effizienter werden können. Wie es in der Motivation vom ersten Kapitel erwähnte wird, dass die Multimedia-Retrieval von dem Semantischen Web nicht nur durch Web-Computer in der Zukunft realisiert wird sondern auch bei allen Web-Geräte wie Mobiltelefon geschaffen werden können. Aber wie kann man denn ein Ontologie-basiertes Multimedia-Retrieval im Mobiltelefon realisieren? Auf jedem Fall direkt mit Ontologie-Query zu spielen ist für den Benutzern nicht praktisch, da die üblichen Leuten hat keine Grundkenntnis von der Ontologie-Query und auch hat keine Lust die komplexe Zeichen zu tippen. Kann das Multimedia-Retrieval direkt mit Multimedia-Daten und Menüfunktionen spielbar?

Die so genannte Bildbasierte Multimedia Retrieval ist nur eine Idee zur Lösung des oben genanntes Problems, welche im Mobiltelefon durch eine GUI-Oberfläche versucht, die Metadaten direkt auf den Bilder von den Mobiltelefonbenutzern abfragen und darstellen zu können und ohne Grundkenntnis von Ontologie-Query zu wissen.

Diese Funktionalität wird als eine Menüoption im Mobiltelefon integriert für den Benutzer bei Durchblättern der Fotogalerie zur Verfügung gestellt (sieh untere Abb.1). Sobald diese Menüoption aktiviert würde, beginnt das Mobiltelefon durch drahtloses Netzwerk die von aktuellem Foto oder Bild zugehörige Metadaten abzufragen. Würde die Metadaten im semantischen Web gefunden, werden die in den Metadata beschriebenen Inhaltelementen unten auf dem Bild gezeigt, welche mit der Tastatur

auswählbar sind. Das ausgewählte Inhaltelement können mit rotem Polygon um seines Profil markiert werden oder auch ein Kommentar daneben gezeigt werden (sieh untere Abbildung).

Abb.1
Beispiel des bildbasierten Multimedia Retrievals

Jedes Inhaltelement hat auch eigene Menü-Funktionen sich zu bearbeiten, zum Beispiel kann der Benutzer mit der Menü-Funktion die auf das aktuelle Inhaltelement beziehende Informationen weiter abfragen, wie Abfrage aller dieses Inhaltelement beinhalteten Bilder oder Abfrage der Einkaufs-information von dem Inhaltelement. Solche Funktionalität ist eine praktische Anwendung von semantischem Web und auch sehr nützlich für den Mobiltelefonbenutzer.

2.4.1 Arbeitsprinzip

In dem Prototypprogramm(Abb.2) wird das RacerPro-System[1]als Eine Semantische Webumgebung um Ontologie-Query zu analysieren und Metadata zu verwalten. Die Bild-Resourcen werden zuerst durch Tboxen und Aboxen, welche mit der Ontologie-beschreibungsprache OWL aufgebaut werden, über die in den Bilder erscheinende Begriffen bzw. die mit den Begriffen konkrete definierte Inhalt-Individuen beschriebt und dann in RacerPro-System hochgeladet, damit sind die Metadaten der Bildes für dem Benutzer in der Internet abfragbar. Wenn der Befehl „Metadata Retrieval“ von dem Benutzer aktiviert würde, erstellt das Handy sofort durch Drahtlos-Netzwerk eine Socket-Verbindung mit dem RacerPro-System und sendet die Ontologie-Query, um alle Metadaten des aktuellen Bildes zu haben. Das RacerPro-System übernimmt die Query von den öffentlichen Porte und sucht die entsprechende Metadaten gemäß der in Query gegebene Bedingungen. Nachdem die Metadaten gefunden wurde, schreibt das RacerPro-System das Ergebnis in einer Zeichenkette und sendet es wieder durch die Socket-Verbindung zurück. Der Kommunikationsvorgang ist in synchroner Weise implementiert und in der Abb.2 dargestellt.

Aber diese von RacerPro-System antwortete Zeichenkette ist häufig nicht von dem Benutzer direkt lesbar und enthaltet viele zusätzliche Symbol und Informationen, die für den Benutzer sinnlos sind. Deshalb muss das Ergebnis vor der Ausgabe gefiltert werden. Die herausgezogene Metadaten werden

[1] RacerPro System ist eine Ontology Reasoning Maschine von Racersystems GmbH & Co.KG.

in einer Datenstruktur gespeichert und dann visuell im Bildschirm dargestellt. Danach kann der Benutzer wie in 2.4 erläuterte Idee die gewünschte Information wählen oder die Informationen über deren Inhalte weiter abfragen.

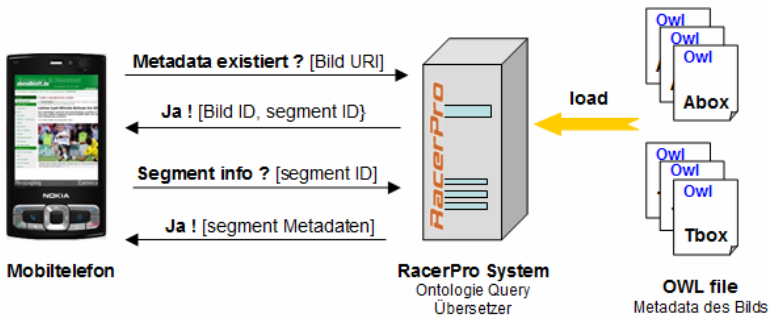

Abb.2
Arbeitprinzip des bildbasierten Multimedia Retrievals

Diese Idee wird in der Studienarbeit durch ein Programm realisiert. Um solches auf graphischen Benutzeroberflächen und visueller Darstellung der Daten basierte Programm effizient zu konstruieren wird das MVC-Paradigma als Entwurfsmuster verwendet, um Bearbeitung und Anzeigen der Daten einfach und leicht zu implementieren.

3 MVC-Paradigma

Bevor ich diese Studienarbeit anfange, wusste ich nur, dass MVC-Paradigma ein Entwurfsmuster für Softwareentwicklung ist. Aber was ist das Entwurfsmuster? Warum benutzen wir Entwurfsmuster?
Wie wird MVC als Entwurfsmuster in Softwareentwicklung verwendet? Wie viele Entwurfsmustern gibt es noch außer MVC? Die Antworten von diesen Fragen werden als der erste Schwerpunkt der Studienarbeit untersucht.

3.1 Geschichte von Entwurfsmuster

In der objektorietierte Softwareentwicklung stoßen die Entwicklern häufig viele Probleme des Entwurfs über wie Programmstruktur, Objektverhalten und Objekterzeugung bei ihrer alltäglichen Arbeiten. Aber häufig gibt es viele verschiedene Entwurfslösungen für ein Problem. Die Entwurfslösung eines Problems kann nicht wieder für andere ähnliche Probleme benutzt werden. Um die Entwürfe einfach wieder verwendbar und leicht änderbar zu ermöglichen werden Entwurfsmustern häufig in objektorientierte Softwareentwicklung verwendet. Jedes Entwurfsmuster beschreibt ein wiederkehrendes Problem und bietet auch den Entwicklern ein allgemeines Konzept an, um das Problem zu lösen.

Im Jahre 1977 und 1979 hatte der Architekt Christopher Alexander in seinen Bücher <<Pattern Language>> und <<Timeless Way of Building>> das Konzept „Muster“ vorgeschlagen, um ein allgemeines Problem über architektonische Qualität zu lösen.

Alexander schrieb:
„Jedes Muster beschreibt eine Aufgabenstellung oder ein Problem, dass sich uns in unserer Umwelt in ähnlicher Form immer und immer wieder stellt. Dabei wird nicht nur das Problem beschrieben, sondern auch die Lösung, so dass diese Lösung millionenfach angewendet werden kann, ohne das Problem zwei mal auf die selbe weise zu Lösen“ ***[DP][EV]***

Aber die Wirkung von dem Konzept im architektonischen Bereich ist weniger als die später in Softwareentwicklung eingebrachte Wirkung. 1987 benutzten Kent Beck und Ward Cunningham das Konzept von Alexander als Entwurfsmuster in Smalltalk-80 verwendet, um die graphischen Benutzeroberflächen zu konstruieren. 1994 brachte Erich Gamma zusammen mit Richard Helm, Ralph Johnson und John Vlissides ein beeinflussendes Buch *„Design Patterns - Elements of Reusable Object-Oriented Software*“**[DP]** heraus, in dem insgesamt 23 Entwurfsmustern ausführlich über ihre Konzepte und Anwendungen in drei Kategorien von *Erzeugende Muster*, *Strukturelle Muster* und *Verhaltensmuster* geschrieben (sieh Abb.3). Es gibt heute auch anderen Entwurfsmustern, die mit

anderen Bedürfnissen von Architektur, Datenzugriff, usw. in Softwareentwicklung verwendet, z.B. MVC für den Entwurf der Architektur.

Erzeugende Muster	Strukturelle Muster	Verhaltensmuster
Abstrakte Fabrik	Adapter	Befehler
Erbauer	Brücke	Beobachter
Fabrikmethode	Dekorierer	Besucher
Prototyp	Fassade	Interpreter
Singletion	Fliegengewicht	Iterator
	Kompositum	Memento
	Proxy	Schablonenmethode
		Strategie
		Vermittler
		Zustand
		Zuständigkeitskette

Abb.3
Klassifikation der Entwurfsmustern

3.2 Konzept von MVC-Paradigma

Die originale Idee von MVC-Pattern war erst mal im Jahre 1978/1979 von norwegische Computer-Wissenschaftler Trygve M. H. Reenskaug bei Fa. Xerox Parc in seinem technischen Dokument***[RT]*** geschrieben, um Entwurf der Software-GUI in Programmiersprache Smalltalk-80 zu benutzen.

Die Idee von Reenskaug war es, dass MVC als eine Lösung für das Problem über Steuerung der großen und komplexen Daten erdacht und es aus drei Komponenten: Model View und Controller beseht. Model ist Kenntnis-Objekt, View ist eine visuelle Darstellung von Model, Controller ist die Link zwischen Benutzer und System. View und Controller werden als die Wergzeuge um das Model visuell zu steuern.

Reenskaug schreibe(von englische Inhalt übersetzt):
*„Das wesentliche Ziel von MVC ist um den Abstand zwischen dem geistigen Modell des Menschen und dem in Computer speicherten digitales Modell überklüften. Die ideale MVC-Lösung bietet dem Benutzer eine Sinnestäuschung, das Modelldaten direkt sehen und bearbeiten zu können. Diese Konstruktion ist sehr nützlich, wenn der Benutzer das gleiche Modell gleichzeitig in verschiedenen Kontexte und/oder von den verschiedenen View-Punkten sehen möchten****[TR]****.“*

Diese Idee war nur 1978/1979 von Reenskaug als ein technisches Dokument geschrieben, danach gibt es keine öffentliche Information mehr über das Konzept von MVC. Bis 1988 war das MVC-Konzept nur erst mal von Glenn E. Krasner und Stephen T. Pope in ihrem öffentlichen Papier***[GE]*** weiter

ausführlich erläutert. Das eingehend und umfassend über die Ausführung und Anwendung der MVC-Konzepts beschriebene Papier***[SB]*** war 1992 von Steve Burbeck öffentlich herausgegeben, in dem wird es erläutert, wie wird MVC als Entwurfsmuster für Applikationsentwicklung in Smalltalk-80 verwendet. Die Aussage über MCV-Konzept von seinem Papier lautet(von englischem Inhalt übersetzt):

*„In der MVC-Paradigma modelliert die Benutzereingaben die Aktion der äußerlichen Welt. Das visuelle Feedback zu dem Benutzer wird deutlich separat mit drei Objekttypen verarbeitet. Das View verwaltet die graphische und/oder textliche Ausgabe und verteilt die Teile des Bildschirms für ihre Anwendung. Der Controller übersetzte die Benutzeraktionen von Maus und Tastatur in Befehle, welche das Model und View steuert, entsprechend zu ändern. Das Model verwalte die Verhalten und die Daten von Anwendungsdomäne und beantworte die von View über seinen Zustand gegebene Abfrage und die von Controller zur Änderung des Zustands gegebene Befehle**[5]**. „*

Zusammenfassung die Aussagen von Reenskaug und Burbeck kann man das Konzept des MVC-Paradigmas mit folgende Folgerung bekommen. MVC-Paradigma ist eine Triade-Struktur, welche aus drei Komponenten Model, View und Controller besteht. Jede Komponente spezialisiert sich auf eine Aufgabe.

3.3 Model

Die Methode zur Konstruierung der Model-Objekte ist im MVC-Paradigma nicht explizit angegeben, MVC teilt den Entwickler nur es mit, wie soll die Model-Objekt verwaltet werden, damit seine Wiederverwendbarkeit und Erweiterbarkeit erhöht werden können. Das Model kann als ein Datenobjekt betrachtet werden, das als ein Struktur die im Programm benutzte Anwendungsdaten speichert und durch verschiedenen Darstellungsformen im Ausgangsrichtung dargestellt werden kann. Außerdem enthält das Model-Objekt häufig auch die zur Bearbeitung der Anwendungdaten benutzten Methoden, die zusammen mit den Daten dem Benutzer eine komplexe Anwendung zur Verfügung stellt.

Zum Beispiel(Abb.4) in dem Windows-System kann der Speicherplatzzustand der Festplatte C als ein Model in drei Darstellungsformen gezeigt werden. Jede Darstellungsform wird als ein View-Objekt betrachtet und zeigt den Benutzern die gleiche Informationen und Funktionen.

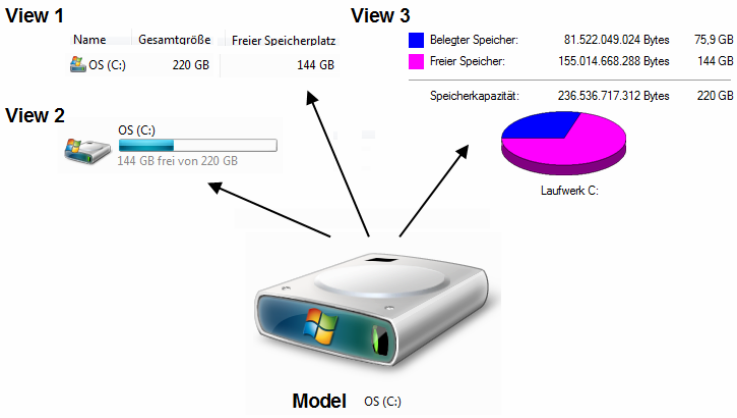

Abb.4
Drei verschiedene View zur Darstellung des Speicherplatzes von Festplatte C.
View 1 Detail-Ansicht; View2 Kacheln-Ansicht; View 3 Prozentverhältnis-Ansicht

3.4 View

Das View-Objekt stellt den Benutzern eine visuelle interaktive Plattform zur Verfügung und bringt dem Benutzer ein Phantasma, die Daten des Model-Objekts direkt sehen und bearbeiten zu können. Es hat zwei führende Aufgaben. Erstens, es bietet den Benutzer visuelle Informationen, welche in Datenobjekte des Model-Objekts gespeichert sind. Zweitens, es bietet den Benutzer die Funktionen, welche die Methoden des Controller-Objekts abbilden, um die gesehenen Informationen zu bearbeiten.

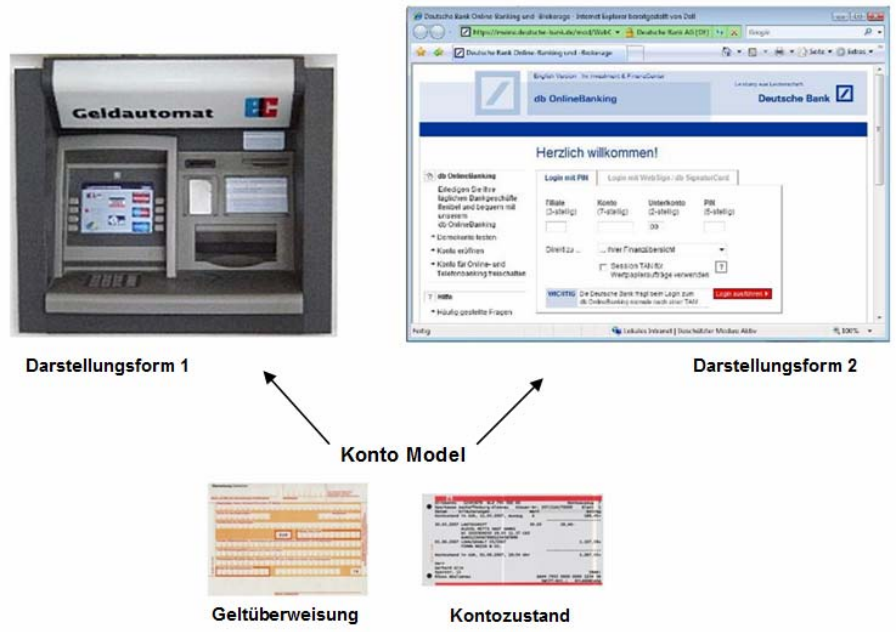

Abb.5

Die Funktionalitäten und Zustanddaten des Konto-Models können
durch Bankautomat und Webseite repräsentiert werden.

Wie in Abb.5 gezeigtes Beispiel kann der Benutzer durch zwei Arte seines Bankkonto verwalten, Bankautomat oder Online Banking. Damit kann der Benutzer sowohl den aktuelle Kontozustand

besichtigen, als auch entsprechende Funktionen wie Geltüberweisung benutzen, welche durch Software-GUI und Webseite dem Benutzer zur Verfügung gestellt werden. Zwei verschiedene Arten repräsentieren die gleiche Funktionalität. Diese aus View-Objekte dargestellten Funktionalitäten werden nicht direkt im View-Objekt aufgebaut, sondern von dem Model-Objekt besorgt.

3.5 Controller

Das Controller-Objekt verwaltet View-Objekte und Model-Objekte, deren Methoden werden nach bestimmte Funktionslogik in verschiedenen Methoden kombiniert, gemeinsam um bestimmte Funktionen zu implementieren und im View-Objekt durch Webseite oder Menübefehle auszugeben. Solange eine Funktion von dem Benutzer durch View-Objekt aktiviert wird, organisiert der Controller sofort die entsprechende Methode von View-Objekte und Model-Objekte, um die aktivierte Ereignis zu bearbeiten. Das heiß, es gibt eigentlich keine Datenverarbeitung im Controller-Objekt, der Controller setzt nur die von View-Objekt übertragte Benutzeranfrage in spezialisierten Aufgaben um und verteilt ihnen zur entsprechende View-Objekte und Model-Objekt. Von dieser Ansicht kann der Controller als „Aufgabeverteiler“ betrachtet werden, er entscheidet welche Model-Objekt und View-Objekt benutzt werden sollte, gemeinsam um die Benutzeranforderung zu erledigen.

3.6 Komponentenverhältnis

Obwohl die Model-Objekte und View-Objekte durch Controller-Objekte getrennt werden, gibt es noch die Kommunikation dazwischen, welche durch Methodenaufruf und aktivierende Ereignisse erfolgen. Untere Abb.6 zeigt das Komponenten-Verhältnis im MVC-Paradigma.

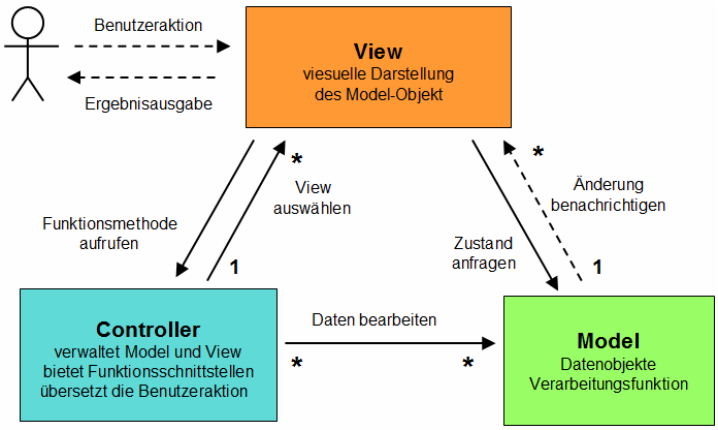

Abb.6
Kommunikation zwischen Model, View und Controller
Durchgezogene Linie zeigt die Methode aufrufen; Gebrochene Linie zeigt die Ereignisse.

Zwischen View und Controller

Controller kann viele View-Objekte verwalten, Jedes View-Objekt ist zur Interaktion zwischen den Benutzer und System verwendet. Es bietet dem Benutzer viele Funktionsmenüs oder Funktionsknopfe in der graphische Benutzeroberfläche, Jede Menüoption und jeder Knopf repräsentiert eine Funktion, die eine Methode von Controller abbildet, und wird vom View-Objekt abgehört. Wird die Funktion auf der graphischen Benutzeroberfläche von Benutzeraktion aktiviert, ruft View-Objekt sofort die entsprechende Funktionsmethode des Controllers, um die von Benutzer geforderte Funktion auszuführen. Nachdem der Controller die Ergebnisdaten von Model erhaltet hat, wählt er den entsprechende View-Objekt die Ergebnisdaten darzustellen. Wobei das View-Objekt es nicht weiß, wo wird die Benutzeranfroderungen bearbeitet werden und woraus die Ergebnisdaten kommen.

View und Controller werden häufig als ein gesamter Teil betrachtet, um eine visuelle Plattform anzubieten, dadurch der Benutzer die Model-Objekt visuell steuern und bearbeiten kann. Aber warum muss die zwei Teile in MVC-Paradigma getrennt werden? Der Grund ist dafür, dass der Entwickler nach eigener Anforderung das Controller-Objekt flexibel austauchen kann, ohne die View-Objekt zu ändern. Damit das View-Objekt und Model-Objekt separat von spezialisierte Entwickler entworfen werden. Um das Ziel zu erreichen wird Entwurfsmuster „Strategie“ verwendet.

Zwischen Controller und Model

Die von Benutzer angefragte Daten oder Funktionen wird tatsächlich nicht in der Methode des Controller-Objekts implementiert, sondern durch weiteren Aufrufen der separaten Methoden von Model-Objekts und View-Objekt erledigt werden. In den Methoden des Controller-Objekts wird nur dieses Funktionslogik zur Organisierung der Model-Objekte und View-Objekte definiert. Das Model ist absolut von Controller entkoppelt, es lässt den Zugriff von Controller auf seine Daten und Methoden zu, aber weiß es gar nicht wofür die Daten bearbeitet und seine Methoden ausgeführt werden.

Zwischen Model und View

Model und View bieten einem 1 zu N Verhältnis, ein View-Objekt darf nur mit genau einem Model-Objekt verbunden werden und ein Model-Objekt kann durch verschiedene View-Objekt dargestellt werden. Das View-Objekt kann direkt den Zustand seins Model-Objekts besichtigen. Wenn die Datenzustand im Model-Objekt sich ändert, muss die abhängige View-Objekte entsprechend aktualisieren, um die gleiche Datenzustand zu halten. D.h. Model-Objekt muss alle abhängige View-Objekte wissen und bei Änderung des Datenzustand ihnen sofort benachrichtigen, damit das Model-Objekt auf den View-Objekte „stark“ bezieht und es auch dazu führt, dass die einzelne Wiederverwendung und Austausch des Model-Objekt unmöglich werden. Tatsächlich ist diese

Synchronisation des Datenzustands nicht direkt von Model-Objekt implementiert, sondern durch Beobachtermuster realisiert.

3.7 Entwurfsmustern im MVC-Paradigma

Obwohl jedes Entwurfsmuster eine Menge ähnliche Probleme selbstständig lösen kann, wird er häufig nicht als eine optimierte Lösung betrachtet. Die Entwurfsmustern sollen verknüpfen, um das Problem effizienter und besser lösen zu können. Folgende Entwurfsmustern sind im MVC-Paradigma behilflich zum Lösen der im Komponentenverhältnis diskutierte Probleme.

3.7.1 Beobachtermuster (Observer)

MVC-Paradigma benutzt Beobachtermuster um den Datenzustand zwischen Model und View zu synchronisieren. Das Beobachtermuster ist ein Verhaltensmuster, welches eine 1 zu N Abhängigkeit zwischen Objekten definiert. Bei Änderungen des Objektzustands werden andre N abhängige Objekte benachrichtigt und automatisch aktualisiert. Es gibt zwei Rollen im Beobachtermuster, ***Subjekt*** und ***Beobachter***. Ein Subjekt kann von einen beliebigen Zahl von abhängigen Beobachtern beobachtet werden. Sobald der Zustand des Subjekts ändert, werden alle Beobachtern benachrichtigt, damit alle Beobachtern den gleichen Zustand wie der Zustand des Subjekts halten. Dieses Verhalten wird auch ***Publisch-Subscribe*** genannt. Das Subjekt ist der Publisher von Änderungsnachrichten. Es weiß nicht, wer seinen Subscriber ist.

Im MVC-Paradigma wird das Model-Objekt als ein Subjekt und seine View-Objekte als Beobachtern betrachtet. Alle View-Objekte müssen eine Interface ***Observer*** einführen, dessen Methodeschittstelle *update( )* wird weiter bei jedem View-Objekt nach der Anwendungsanforderung aufgebaut. Danach werden alle von dem Model-Objekt abhängige View-Objekte durch der Methode *registerObserver (Observer o)* als die Observer-Objekte im Model-Objekt angemeldet bzw. „subscribe“ die Änderungnachrichten(sieh Abb.7). Sobald den Zustand des Model-Objekt ändert, wird das Model-Objekt die gemeinsame Methodeschnittstelle *update( )* von allen im sich angemeldeten Observer-Objekte implementieren, um die Änderungen zu aktualisieren. Dieser Algorithmus wird bei *Notify( )* konstruiert. Aber Model weiß nicht welche View-Objekte im sich angemeldet werden und wie deren *update( )* Methode aufgebaut werden. Damit das Model-Objekt komplett von dem View-Objekt entkoppelt wird und einzelne Bearbeitung der Model-Objekts keine Einwirkung von View-Objekt hat.

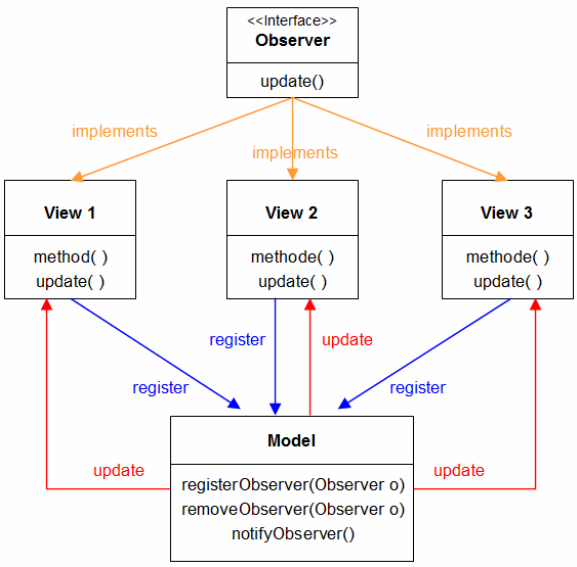

Abb.7
Beobachtermuster im MVC

3.7.2 Stragetiemuster (Stragegy)

View und Controller implementiert häufig das Strategiemuster um das Controller-Objekt austauchen zu können. Das Konzept des Stragegiemusters ist von Erich Gamma im seinen Buch***[DP]*** so geschrieben,

„Definiere eine Familie von Algorithmen, kapsele jeden einzelnen und mache sie austauschbar. Das Stragegiemuster ermöglich es, den Algorithmus unabhängig von ihn nutzenden klienten zu variieren“

In objektorietierter Programmierung werden unterschiedliche Algorithmen mit Klassen gekapselt, jede solche Klasse heißt eine Stragetie und hat eine abstrakte Superklasse, welche die gemeinsamen abstrakten Funktionsschnittstellen definiert. Zum Beispiel(sieh Abb.8), Controller ist eine abstrakte Klasse, welche nur die abstrakten Funktionsschnittstellen einer Strategie-Familie definiert. Die Klasse Controller haben drei Subklassen, die alle Funktionsschnittstellen vom Controller vererben, in der unterschiedlichen Strategie konstruiert werden, um unterschiedliche Aufgaben zu erledigen. Im View-Objekte wird die abstrakte Klasse Controller als eine private Variable deklariert, jedes von View gefangene Benutzerereignis wird eine Funktionsschnittstelle des Controllers abgebildet. Dadurch kann das View-Objekt eine beliebige Strategie von den Subklassen des Controllers benutzen.

Der Vorteil von dieser Struktur liegt darin, dass die Entwürfe von View-Objekte und Controller-Objekte nach die Funktionsschnittstellen von der abstrakten Klasse separat durchgeführt werden können. Bei Austausch der Strategie gibt es keine Wirkung für View-Objekte.

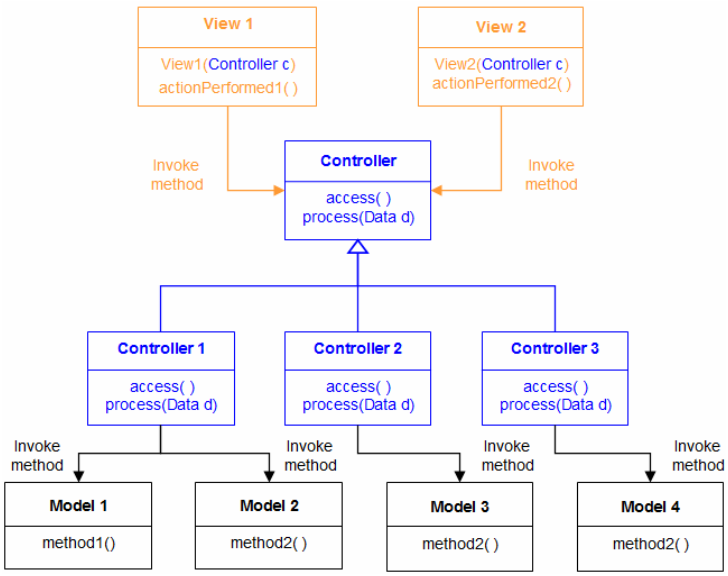

Abb.8

Strategiemuster im MVC

3.8 Vorteile und Nachteile des MVC-Paradigmas

Vorteile des MVC-Paradigma sind sehr selbstverständlich:

- Deutliche Architektur durch Dekomposition der Programmcode nach der Aufgabeverteilung in drei Komponenten, Anwendungsfunktionen(Model), graphische Benutzeroberfläche(View) und Arbeitslogik(Controller).
- Durch solche Aufgabentrennung kann die Applikation von verschiedene Entwicklern, jeder auf eine Komponente spezialisiert, zusammen konstruiert werden.
- Mit Anwendung des MVC-Paradigma wird Austausch und Änderung der einzelnen Komponente ermöglicht und vereinfacht. Z.B. für die Änderung der Benutzeroberfläche braucht nur das entsprechende View-Objekt ändern, nicht auf andere zwei Komponenten zu beziehen.
- Wiederverwendung des Model-Objekts ist möglich. Damit es Wiederholte Arbeiten vermeidet und Kosten spart.

Obwohl MVC-Paradigma viele Vorteile hat, gibt es aufgrund des verteilten Programmcode auch viele Nachteile,

- MVC-Paradigma vergrößert das Code-Volumen. Da zusätzliche Kommunikationen zwischen drei Komponenten und Synchronisation des Model-Objekts viele Code und Kosten verbraucht.
- Trennung der Programmcode führ zu hoher Komplexität. Damit reduziert die Effizienz der Programmentwicklungs.
- Das View-Objekt und Controller-Objekt sind sehr enge verbunden. Austausch und Änderung des View-Objekts werden aufgrund des Strukturentwurfs des Controller-Objekts beschränkt.

- Die konkrete Methode zur Aufbau jeder Komponenten wird nicht im MVC-Paradigma explizit angegeben, es erhöht die Schwierigkeit zum Verstanden und Anwenden des MVC-Paradigma in der Programmentwicklung.
- Schwierige Verwendung von MVC mit modernen Werkzeugen für Bedienoberflächen.

4 J2ME

4.1 Allgemeine

Im Jahre 1999 Juni gab Sun Microsystem J2ME(Java 2 Micro Edition) aus, welche eine Programmiersprache von Java-Familien ist, um die kleine Applikation von den Geräte, deren Stromquelle und Netzwerkverbindung sowie GUI-Entwicklungsfähigkeit beschränkt werden, wie Mobiltelefon und PDA, zu entwickeln.

4.2 Architektur

Die J2ME-Technologie basiert auf drei Elementen: ***Konfiguration***, ***Profile*** und ***optionales Packet***.

- Die Konfiguration bietet die meisten grundlegenden Bibliotheken und virtuelle Maschine zum Aufbau der Java-Laufzeitumgebung an und wird gemäß des Hardwaretyps zwei Kategorien unterscheidet, Connected ***Limited Device Configuration(CLDC)*** und ***Connected Device Configuration(CDC)***.
- Das Profile ist ein komplettes Set von APIs für Programmentwicklung aufgrund der Konfiguration. Z.B. Foundation-Profile basiert auf CDC und Mobile Information Device Profile (MIDP) basiert auf CLDC.
- Das optionale Packet ist eines Set von Technologie-spezifizierte APIs.

Das folgende Bild Abb.9 zeigt die Komponenten von Java ME Technologie und die Relation von anderem Java Technologies.

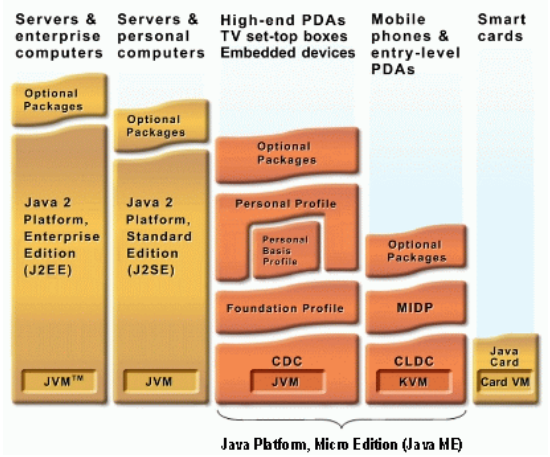

Abb.9
Architektur der Java-Technologies-Serien ***[SMj]***

4.3 CLDC

Für Mobiltelefon werden Connected Limited Device Configuration(CLDC) und Mobile Information Device Profile(MIDP) angewendet.

CLDC definiert mit der grundlegende APIs und eines ***Kilobyte virtuelles Maschine(KVM)*** die kleinstmöglichen Konfiguration einer J2ME-Laufzeitumgebung für Hardware-beschränkte Geräte, wie Mobiltelefon, die Anwendungsapplikationen werden weiter von MIDP definiert. CLDC sind zu den Prozessoren von 16Bit und 32Bit mit Hauptfrequenz von mehr als 16MHz und Speicher weniger als 512KB passt. Zurzeit braucht das JVM von Windows-System mindestens 16MB. Die Spezifikation von der ersten Version CLDC1.0***[SMc]*** kann man in Webseite von Fa. Sun-Microsystem finden. Außer CLDC1.0 gibt es auch CLDC1.1, die zum Vergleich mit CLDC1.0 mehr Leistungsfähigkeit, wie z.B. Die Gleitkommaberechnung, ergänzen.

4.4 MIDP

MIDP stellt aufgrund CLDC eine komplette Java-Plattform für Entwicklung der Applikationen in Mobiltelefon zur Verfügung. In MIDP besteht folgende API-Packet,

- ***javax.microedition.lcdui*** *(User Interface API)*
  Es bietet für Anwendung des MIDPs ein komplettes Set APIs zum Aufbau des Benutzerinterfaces an.
- ***javax.microedition.rms*** *(Persistence Package)*
  Es bietet einen Mechanismus für MIDlets an, um Daten andauernd zu speichern und später wieder-zugewinnen.
- ***java.microedition.midlet*** *(Application Lifecycle package)*
  Es definiert alle MIDP-Applikationen und die Interaktion zwischen den Applikationen und ihrer Laufumgebung.
- ***java.microedition.io*** (Networking package)
  Es definiert die Klassen für generische Netzverbindung-Rahmen.

Es existiert bisher das MIDP1.0 und MIDP 2.0. Das MIDP ist nach Unten verträglich, die MIDP1.0-Applikationen können in MIDP2.0-Mobiltelefon ausführen. Es gibt die Volumentbeschränkung in MIDP1.0, die Softwaren dürfen nicht groß als 64K sein.

4.5 MIDlet

Im Vergleich mit Applet von Web-Client und Servlet von Server heißt die Applikation von MIDP ***MIDlet*** . Jede MIDlet besteht aus mindestens ein Klasse, die auf die MIDP ausgeführt wird und muss die abstrakte Klasse *java.microedition.midlet.MIDlet* vererben, welche drei abstrakte Funktionsschnittstellen ***startApp( )***, ***pauseApp( )*** und ***destroyApp( )*** definieren, um die drei Zustände des MIDlets konstruieren. Der entsprechende Zustandwechsel wird von ***Application Management Software(AMS)*** des Endgerätes gesteuert werden.

Wie in Abb.8 gezeigt, Nachdem Konstruktor aufgerufen wurde, wird ein Objekt des aktuellen MIDlets erzeugt. Das MIDlet beginnt von *startApp( )*, diese Methode wird ausgeführt sowohl bei Initialisierung der MIDlet als auch nach Aktivierung von „Pause“ Zustand. Die Methoden *pauseApp( )* und *destroyApp( )* lassen die Applikation zu „Pause“ bzw. „Destroyed“ Zustand eintreten. Der Zustandswechsel wird nicht immer von der AMS veranlasst, in den meisten Fällen wird er sogar vom MIDlet selbst initiiert. Die Abb.10 zeigt, welche Methoden von der AMS und welche von dem MIDlet aufgerufen werden, um zwischen den einzelnen Zuständen zu wechseln.

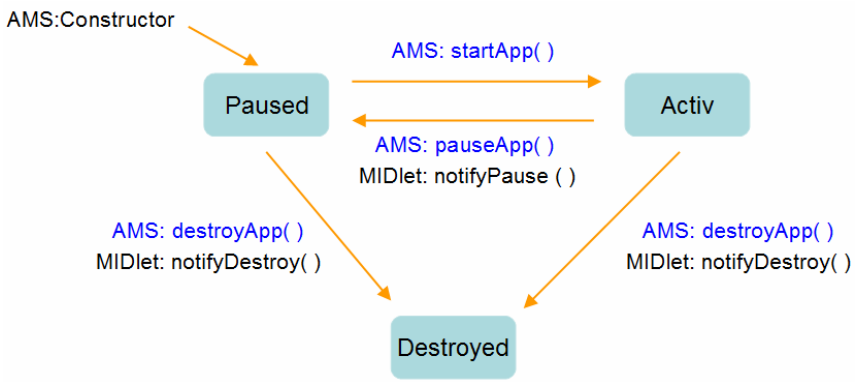

Abb.10
Zustandsübergänge eines MIDlets

Beispielweise zeigt folgender Code eine Text „Hello World !“ mit 15 Größe und Titel von „Hello MIDlet“ in Display des Mobiltelefons. Der Textinhalt wird als eine visuelle Komponente betrachtet und im Konstruktor mit eine Menüoption „Exit“ zusammen konstruiert. Die in der Methode *startApp( )* geschriebene Anweisung bedeutet es, dass bei Programmstarten wird die visuelle Komponente *tb* im aktuellen Bildschirm ausgegeben. Die letzte Methode definiert alle Algorithmen, welche die Benutzerereignisse bei Aktivierung der Menüoptionen bearbeiten, z.B. bei Aktivierung der „Exit“ Menüoption ändert das Programm druch den *destroyApp( )* seinem Zustand im Zerstörung.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
        private Command exitCommand;
        private TextBox tb;

        public HelloWorld(){
                exitCommand =new Command("Exit",Command.EXIT,1);
                tb =new TextBox("Hello MIDlet","Hello,World!",15,0);
                tb.addCommand(exitCommand);
                tb.setCommandListener(this);
        }

        protected void startApp(){
                Display.getDisplay(this).setCurrent(tb);
        }

        protected void pauseApp(){ }

        protected void destroyApp(boolean u){ }

        public void commandAction(Command c,Displayable d){
                if (c ==exitCommand){
                    destroyApp(false);
                    notifyDestroyed();
                }
        }
}
```

4.6 Anzeige und Visuelle Komponenten

Das MIDlet ist durch eine Instance der Klasse ***Display*** dem Bildschirm des Mobiltelefons zu steuern. Jedes MIDlet hat nur eine einzige Instance der Klasse Display. Alle in Bildschirm gezeigte visuelle Komponenten müssen in dem Objekt der Subklasse der Klasse ***Displayable*** konstruieren und dann werden in dem Objekt der Klasse *Display* repräsentiert. Die Klasse *Displayable* hat zwei Subklassen ***Screen*** und ***Canvas***, welche zur Implementierung der High-Level-Anzeige bzw. Low-Level-Anzeige in J2ME verwendet. Wie Abb.11 dargestellt, dass die von Klasse *Canvas* und Screen aufgebaute Instance der Klasse *Displayable* durch der Methode *set.Current (Displayable)* von der Klasse *Display* in Bildschirm des Mobiltelefon repräsentiert wird.

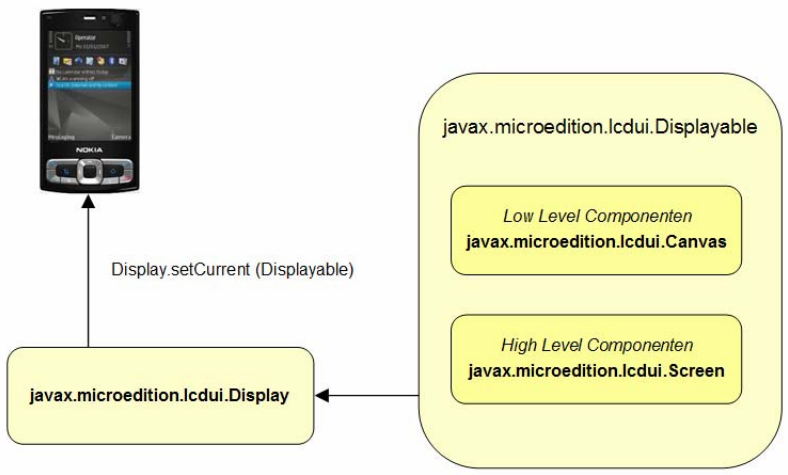

Abb.11

Alle visuelle Komponenten sind durch die Subklassen von der Klasse *Displayable* in zwei Level konstruiert und dann durch die Methode *setCurrent(displayable)* von Klasse Display werden die visuelle komponenten in dem Bildschirm des Mobiltelefon gezeigt.

4.6.1 High-Level Anzeige

Die Subklasse ***Screen*** ist für die High-Level Anzeige verwendet. Die so genannte High-Level Anzeige heißt, dass die in Bildschirm gezeigte visuelle Objekte z.B. List , Druckknopf bereits als standarde komponenten vom Entwickler in den Subklassen der Klasse *Screen* konstruiert, z.B.in Abb.12 dartestellte Subklasse ***Alert***, ***Form*** und ***Iterm*** definieren die visuelle Objekte von Hinweisfenster, Komponent-behälter bzw. Komponenteneigenschaft. Es gibt auch einige Subklassen von der Klasse *Screen* für andere Anwendung definiert, deren Methoden und Anwendungsangaben kann man in J2ME-API finden.

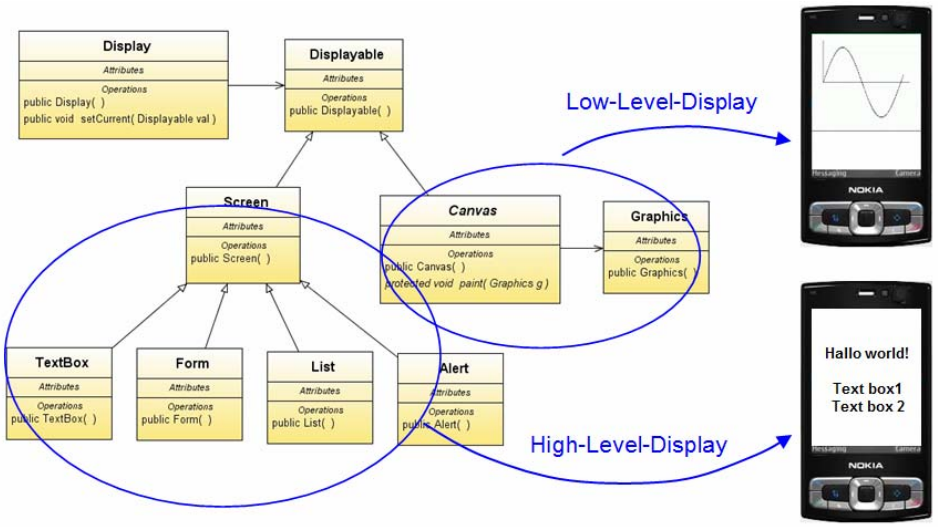

Abb.12
Architektur der Subklassen von der Klasse *Displayble*

Mit solchen fertig entwicklten standarden Komponenten können der Programmentwickler einfach eine GUI im Mobiltelefon erstellen, aber diese standardisiere Komponenten können nicht zu allen Anwendungen anpassen, sondern sich nur für die einfache GUI-Strukturen geeignet.

4.6.2 Low-Level Anzeige

Im Vergleich mit Hight-Level Anzeige werden in der Low-Level Anzeige keine standarde visuelle Komponenten verwendet, alle Komponenten müssen vom Programmentwickler nach den praktischen Anforderungen selbst konstruiert werden.

Die Low-Level Anzeige wird in J2ME durch der Klasse ***Canvas*** *und* ***Graphics*** realisiert und ist häufig in Spielprogrammierung und graphische Bearbeitung im Mobiltelefon verwendet. Das Objekt von Klasse *Canvas* kann als ein Maltuch beobachtet werden, in dem kann man alle zweidimensionale geometrische Graphen und auch Text malen. Die Klasse *Graphics* ist ähnlich wie ein Malzeug, das zur Konstruktion der visuelle Komponenten im Objekt der Klasse *Canvas* benutzt wird, und definiert viele Methoden die für z.B. Farben, gerade Linien, Kurven usw. darstellen zu können.

In der Klasse Canvas gibt es eine abstrakte Methode *paint(Graphics g)* zur Implementierung aller Malmethoden vom Objekt der Klasse *Graphics* und alle Objekte der Klasse *Canvas* müssen diese Methode vererben und umschreiben.

4.7 MVC-Paradigmas in J2ME

Aufgrund den Kenntnisse von Abschnitt 3.6 ist es selbstverständlich, das View-Komponent in J2ME durch den Subklassen der abstrakte Klasse *Displayable* konstruiert wird. Jede innerhalb eines Programmlaufs benutzte Benutzeroberfläche wird als ein View-Objekt betrachtet, um verschiedene Anwendungen darzustellen. Die in Bildschirm gezeigte Anwendungsdaten und Anwendungs-funktionen werden in den Model-Objekte konstruiert. Alle Anzeigewechseln und Funktionslogik werden durch Controller-Objekt implementiert.

Es gibt einige mögliche Architekturen, welche MVC-Paradigma verwenden, um Applikation des Mobiltelefons mit J2ME zu entwickeln.

4.7.1 Single Controller MVC-Paradigma

Wie Abb.13 zeigt, es gibt nur ein Controller-Objekt für alle View-Objekte und Model-Objekte zu verwalten. Jedes View-Objekt repräsentiert eine Benutzeroberfläche und stellt ein bestimmtes Daten-Objekt dar, welches im Model-Objekt konstruiert wird. Alle von View-Objekte gefangene Ereignisse und alle im Model-Objekte bearbeitete Daten, sowie dazwischen liegende Anzeigewechsel werden im einzigen Controller-Objekt bearbeitet(sieh Abb.14).

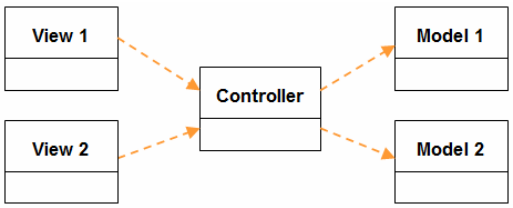

Abb.13
Single Controller MVC-Paradigma

Solche Struktur hat einen einfachen Programlauf und wenige Komplexität. Aber wenn das Programm viele Benutzeroberflächen hat, wird dieses Controller-Objekt sehr gross geworden und alle Funktions-logike werden aufeinander in einer Klasse gemischt. Dazu ist Änderung und Austausch der einzelne Funktion für den Entwickler nicht einfach und Effizienz der Programmentwicklung wird reduziert. Diese Struktur eignet sich nur für wenige Benutzeroberflächer besitzte Applikationsentwicklung.

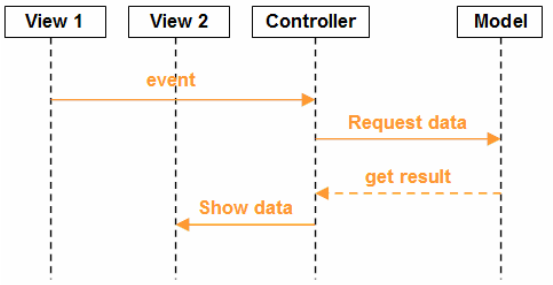

Abb.14
Anzeigewechsel bei einzelne Controller MVC-Paradigma

4.7.2 Multiple Controller MVC-Paradigma

Wenn die Applikation viele View-Objekte und Model-Objekte hat, wird eine Multipel-Controller-Architektur verwendet. Model-Objekte und View-Objekte können nach Funktionalität der Applikation gruppiert und separat durch unterschiedlichem Controller-Objekt verwalten(sieh Abb.15). Die Kommunikation zwichen den Funktionsgruppen wird durch Controller ausgeführt. Programmstruktur wird dadurch mehr sauberer und deutlicher. Aber bringt diese Architektur der Applikation viele

Kommunikations-aufwand und höhere Komplexität und ist für kleine J2ME-Applikationsentwicklung nicht geeignet.

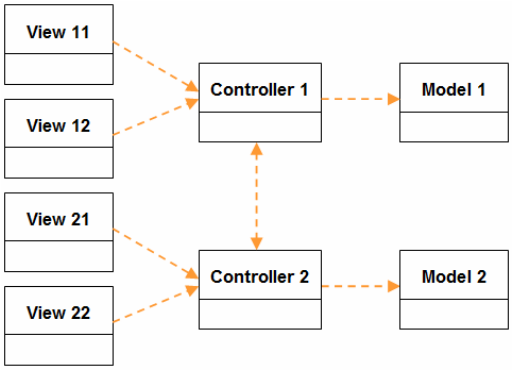

Abb.15
Multipel Controller MVC-Paradigma

4.7.3 Vereinfachte MVC-Paradigma

Oben erwähnte zwei MVC-Paradigma sind meistens in PC-Umgebung für grosser Applikations-entwicklung verwendet und für kleine J2ME-Programmentwicklung nicht angepasst, da die Hardware des Mobiltelefon wegen kleines Volumen beschränkt wird, höhe Komplexität führt zu schlechte Leistung. Dazu kann MVC-Paradigma angemessen verändert werden. Abb.16 zeigte ein vereinfachtes MVC-Paradigma, wobei das View-Objekt und Controller Objekt in einer gesamter Teil integrieren. Jedes View-Objekt wird durch eingnem Controller-Objekt verwaltet und das ganz View-Controller-Objekt repräsentiert eine Benutzeroberfläche und stellte ein Model-Objekt dar. Diese Architektur koppelt zwar die View und Controller, aber reduziert sie die Kommunikationsaufwand zwischen View-Objekt und Controller-Objekt. Dieses vereinfachte MVC-Paradigma wird für die Applikation, deren graphische Benutzeroberflächen nicht komplex ist, sehr geeignet.

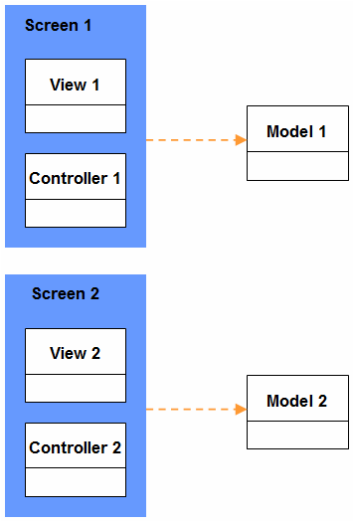

Abb. 16
Vereinfachte MVC-Paradigma

5 Entwicklungsumgebung für Mobiltelefon

5.1 Nokia Mobiltelefon

Nokia N95 wird als die Hardwareumgebung in der Studienarbeit benutzt. Es basiert auf Symbian OS v9.2 Betriebssystem und mit 320x240 pixel Bildschirmsauflösung ausgestattet. Deren Applikationen sind mit den Nokia-Entwicklungsplattform von S60 3rd Edition, Feature Pack 1, CLDC 1.1， MIDP 2.0 Entwickelt.

Abb.17
Nokia Mobiltelefon N95

Symbian OS is ein offenes Mobiltelefon-Betriebssystem, das von dem Fa. Symbian Ltd speziell für moderne Smartphone von 2G, 2.5G und 3G entwickelt wird und von vielen führenden Mobiltelefon-hersteller lizensiert wird.

5.2 NetBeans IDE

Es gibt zurzeit zwei wichtige Entwicklungsumgebung für J2ME, NetBeans IDE und Eclipse. In dieser Studienarbeit benutzen wir NetBeans IDE 6.0 Mobility als Entwicklungsumgebung.

Die NetBeans IDE ist eine Entwicklungsumgebung und als ein Werkzeug für Programmierer, um Programme zu schreiben, testen, debuggen, profilieren, usw. Sie ist ein mit java geschriebenes Open Source Projekt und unterstützt aber jede Programmiersprache. NetBeans 6.0 Mobility ist eine spezielle Version für Mobilegeräte und stattet fast alle Werkzeuge aus, die für Entwicklung der mobile Applikation verwendet werden. Die neueste Version NetBeans Mobility 6.0 kann man unter der Link http://www.netbeans.org kostenfrei unterladen und installieren.

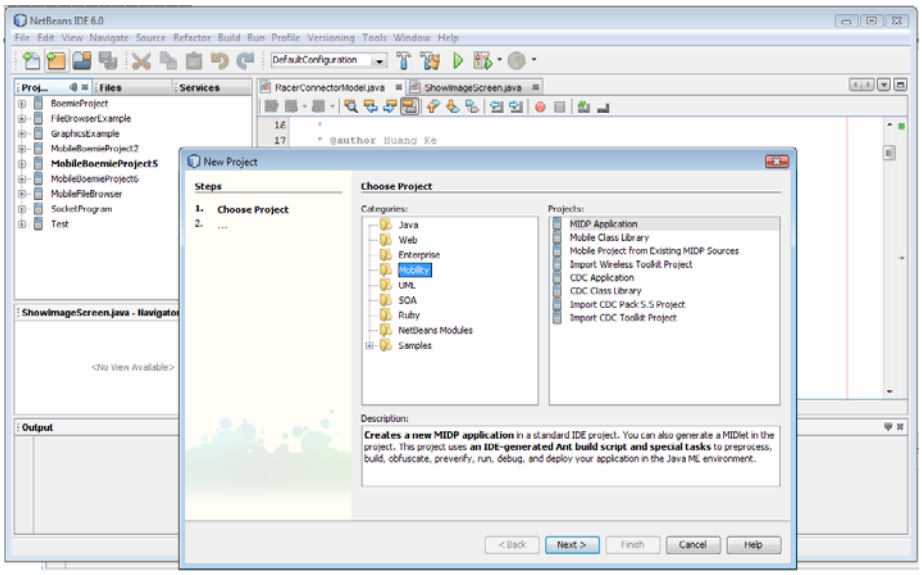

Abb.18
NetBeans IED 6.0 Benutzeroberfläche

NetBeans IDE stellt außer MIDlet auch einen Datentyp von visuelle MIDlet für visuelle Programmierung zur Verfügung. Mit Erstellung der Daten von visuallen MIDlet wird Netbeans entsprechende Palette geladen, die bei der rechten Seit neben der Codebearbeitungsfenster befindet (Abb.19) und viele Komponenten für High-Level Anzeige enthält. Die Programmentwickler können damit einfach die Komponent von Palette Panel die standard Komponent und auch so wie Steuerungsbefehler in Bildschirmfläche einziehen, die entsprechende Code wird automatische in dem Codebearbeitungs-fenster generiert.

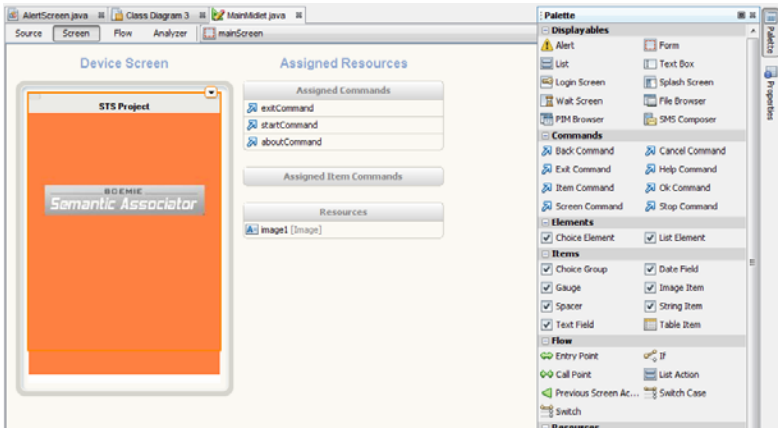

Abb.19
Palette Panel und Bearbeitungsfenster der Visuelle MIDlet

Außer die Codebearbeitung und visuelle Komponentbearbeitung kann die Sequenz des Anzeige-wechsels aller Benutzerinterface auch visuell in *Flow* Fenster bearbeitet werden (Abb.20).

Obwohl die visuelle MIDlet von NetBeans IDE wie andere visuelle Programmierung viele Vorteile hat, ist sie für spezielle Anwendung und individuelle Entwürfe der Programm nicht praktisch und geeignet.

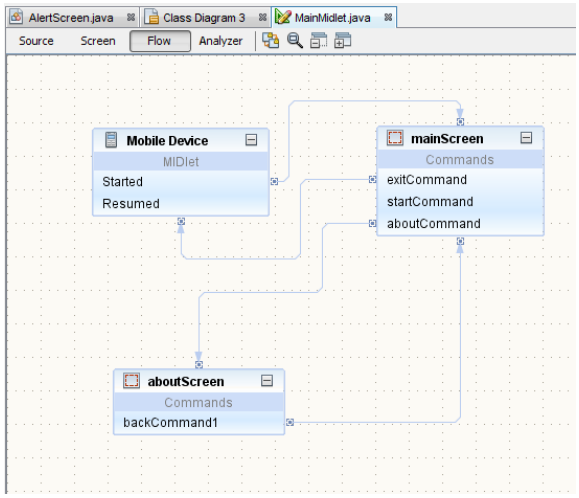

Abb.20
Flow Fenster für Bearbeitung des Anzeigewechsels

5.3 Mobiltelefon-Emulator

Nachdem das Programm fertig konstruiert ist, soll das Programm vor der Installation getestet werden. Emulator wird zum Test die Laufzeitumgebung der Mobilteleon simuliert. Jeder Hersteller des Mobiltelefons hat eigenen Emulator für mobile Applikation zu testen. Hierbei wird zwei Emulatoren vorgestellt, welche in der Studienarbeit benutzt werden.

5.3.1 S60 Entwicklungsplattform

S60-Plattform ist ein Softwareentwicklungssystem (Software Development Kit, SDK) für Smartphone und von Nokia Mobile Software entwickelt. Es basiert auf Symbian OS Betriebssystem und simulierte eine reale Laufzeitumgebung für Entwicklung der Applikationen im Nokia Mobiltelefon zu verwenden. S60-Plattform bietet die Entwickler besondere Ausgabe und Merkmalparket (feature pack) für unterschiedenen Mobiltelefons an. Z.B in dieser Studienarbeit wurde Nokia N95 der S60-Plattform von Edition 3 und Feature Pack 1 verwendet. Die untere Abb.21 zeigt die Architektur der S60-Plattform, die aus *„User Interface Style“*, S60 Applications, S60 Application Services, S60 Java™ Technology Services und S60 PlatformServices und Symbian OS Extensions besteht und auf Symbian OS Betriebssystem und grundlegende Hardwaren befindet.

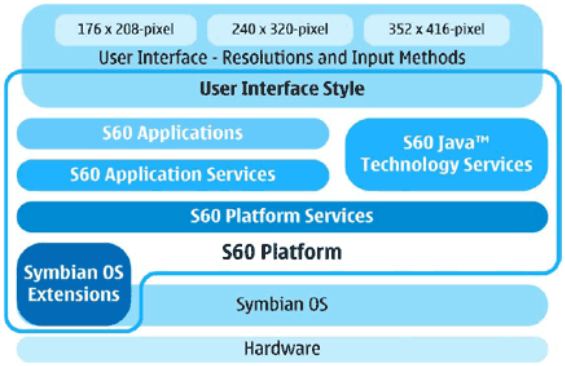

Abb.21
Architektur der S60-Plattform (Resource von wiki.forum.nokia.com)

Die S60-Plattform lass sich in dem Nokia Forum***[NF6]*** nach Anmeldung kostenfrei unterladen zu können. Nachdem die NetBeans IDE in dem Computer installiert wird, kann man einfach, wie Abb.22 zeigt, im Menupunkt „Tools“ mit Funktion „Java Platform“ die S60-Plattform in NetBeans IDE hinzufügen.

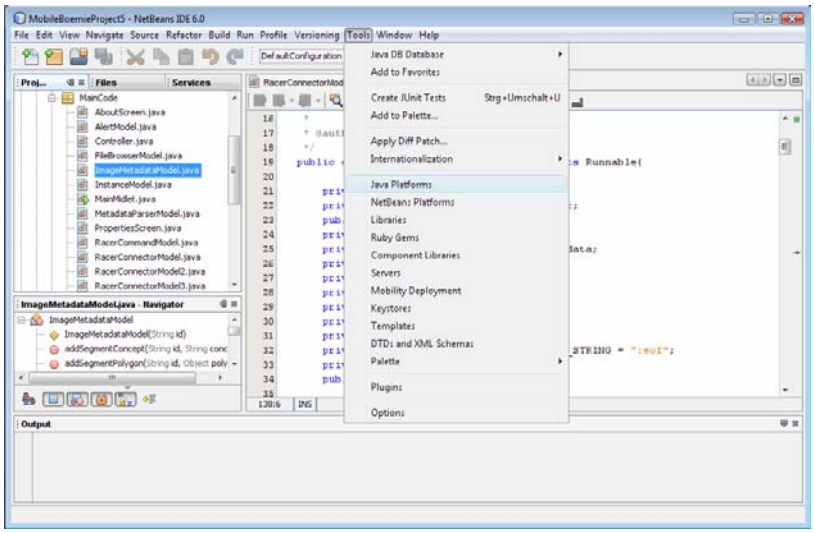

Abb.22
Hinzufügung der S60-Plattform in NetBeans IDE

5.3.2 Wireless Toolkit (WTK)

Das Java Wireless Toolkit (WTK) ist eine Werkzeugpaket von Fa. Sun Microsystem für Entwicklung der J2ME-Applikation, welche auf Connected Limited Device Configuration(CLDC) und Mobile Information Device Profile(MIDP) basiert, verwendet. Im Vergleich von S60 bietet das WTK nur eine grundlegende universale Laufzeitumgebung für unterschiedliche Mobiltelefons, um die Applikation, die auf CLDC/MIDP kompatiblen mobilen Endgeräten lauffähig sind, zu entwickeln.

In NetBeans IDE 6.0 ist bereits die WTK 2.5.2 integriert. In der Eigenschaft des Projekts (Abb.23) von NetBeans IDE kann der Entwickler nach eigenen Bedürfnissen die Emulatoren wählen, um die Mobiltelefone anpassen zu können.

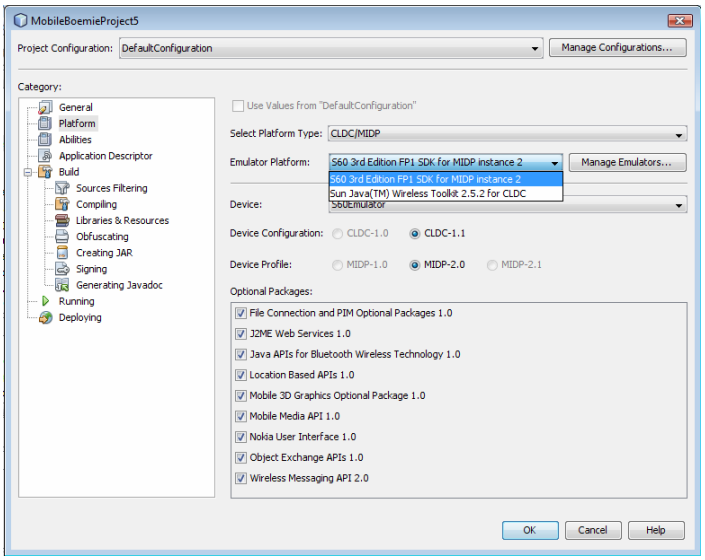

Abb.23
Einstellung des Emulator in NetBeans IDE

5.4 Installation des MIDlets in Mobiltelefon

Sobald das MIDlet an einem Emulator gelaufen war, wird bei der nächste Schritte das MIDlet an dem mobilen Endgeräte gelaufen. Installation des MIDlet in Endgeräte wird durch AMS (Application Manage Software) implementiert. Es gibt einige Arten zur Installation des MIDlets. Auswahl der Methode hängt meistens von der Verfügbarkeit der vorliegenden Hardwaren. Die führende Methoden bei der Programmentwicklung sind: Over-the Air(OTA), Installation durch serial cable, Bluetooth, Email und MMS. In der Studienarbeit wird mit OTA das MIDlet in Nokia-Mobiltelefon installiert.

5.4.1 Over-the-Air (OTA)

Mit Einsatz von OTA wird das MIDlet zuerst an einem Webserver installiert und dann mittels des Internet-Microbrowser in den mobilen Endgeräte unterladet. Eine ***JAR-Datei*** und eine ***JAD-Datei*** sind bei OTA-Installation erforderlich.

Solange ein J2ME-Projekt erstellt wird, besteht es aus zwei Daten, eine JAR-Datei und eine JAD-Datei. In NetBeans IDE befinden sich die zwei Daten in dem Verzeichnis *project folder/dist/* . JAR-Datei enthaltet alle kompilierte Programmcode sowie Mediaresourcen von Bilder, Audio Video usw. JAD-Datei ist eine Text-Datei, die enthältet die Beschreibung über MIDlet, durch Lesen JAD-Datei kennt AMS das MIDlet.

Ausführungsschritten:

1. Kopiert die JAR-Datei und JAD-Datei des MIDlets zu dem Wurzelverzeichnis des Webservers z.B. webapps\ROOT von Tomcat.

2. Erstellt dann eine WML-Datei zusammen mit der JAR-Datei und JAD-Datei in dem gleichen Wurzelverzeichnis, um die JAD-Datei bzw. das MIDlet in Internet veröffentlichen zu können. WML(Wireless Markup Language) ist eine XML-Sprache, die einer Document Type Definition (DTD) unterliegt, und ist auch Teil des Wireless Application Protocal (WAP). Zum Beispiel:

   Dateiname: **download.wml**

```
<?xml version="1.0"?>
  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
  <wml>
  <card title="welcome" id="main">
  <p>MIDlet download</p>
  <p align="left"><a href="MyMobileApplication.jad"></a></p>
  </card>
  </wml>
```

3. Konfiguriert die Internet Media Type (MIME-Typen) des Webservers. Der MIME-Typ von der JAD-Datei wird mit den Wert *„text/vnd.sun.j2me.app-descriptor“* zugewiesen und der MIME-Typ von der JAR-Datei wird mit den Wert „ *application/java-archive“* zugewiesen.

   Oder prüft die *conf\web.xml* von Tomcat, ob diese Datei folgende Information hat, falls nicht, addiert diesen Code zu der web.xml Datei.

```
<mime-mapping>
  <extension>jad</extension>
  <mime-type>text/vnd.sun.j2me.app-descriptor</mime-type>
```

```
</mime-mapping>
<mime-mapping>
  <extension>jar</extension>
  <mime-type>application/java-archive</mime-type>
</mime-mapping>
```

4. Schaltet das Mobiltelefon ein. versichert das Mobiltelefon mit Internet zu verbinden. Öffnet einen WAP-Browser und gibt die absolute URL Adresse von wml-Datei ein, die in Webserver befindet, z.B. http://webserver.com/download.wml. dann tritt auf der geöffnete Webpage die zwei Reihe Wörte „Welcome“ und „MIDlet download“, die in wml-Datei definiert werden. Klickt das Link „MIDlet download“.

5. Solange das Link von wml-Datei geklickt wird, wird die Application Management Software (AMS) aktiviert und zeigt der Name, Version Ursprung und entsprechende Information von MIDlet. Bestätigt und dann das MIDlet wird installiert.

6. Das MIDlet wird nach Installation automatisch ausgeführt, oder findet man es in der Programmsmenü des Mobiltelefons manuell zu starten.

Für weitere Information über OTA-Installation kann man in Nokia-Forum***[NF]*** finden. Unter anderem, die zwei Dokumenten ***[NFo]*** und ***[NFc]*** sind verfügbar.

6 Entwurf des Programms mit MVC-Paradigma

Aufgrund der Grundkenntnisse von MVC-Paradigma und J2ME aus den vorherigen Kapiteln wird nun ein Programm entworfen, um die in 5.4 erwähnte Idee zu realisieren und MVC-Paradigma in der Praktischen J2ME-Programmentwicklung zu verwenden. Das Programm basiert sich mit NetBeans IDE als Entwicklungsumgebung entwickelt und die entsprechende reale Ausführungsumgebung des Mobiltelefons wird durch WKT und S60-Plattform simuliert. Architektur.

6.1 Analysis des Programmablaufs

Wie in Abschnitt 5.4 geschrieben ermöglicht das Programm die Metadaten des Bildes im Mobiltelefon abfragen zu können. Um das Programmentwurf auf MVC-Paradigma beziehen zu können wird zuerst das Aktivitätsdiagramm für Programmablauf analysiert.

Das Abb.17a zeigt ein von der Benutzeransicht aufgebautes Aktivitätsdiagramm, welche die führenden Aktivitäten von den Benutzern im Programmablauf darstellt. Nachdem das Programm startet, sieht der Benutzer die erste Benutzeroberfläche, in der die führende Funktionen und Informationen des Programms gesammelt sind. Wird die Menüoption „local picture“ ausgewählt, wird das Bildschirm zur Benutzerfläche gewechselt, die alle im Mobiltelefon gespeicherte Bilder in einer verkleinerten Form zeigt, damit der Benutzer die Bilder auswählen kann. Wird ein Bild ausgewählt und mit „Ok“ Druckknopf bestätigt, wird das gewählte Bild in neuer Benutzeroberfläche geöffnet und in vergrößerter Form gezeigt. Durch die Menüoption kann der Benutzer weiter deren Metadata abfragen. Sobald das Metadata gefunden wird, wird es auf dem abgefragten Bild gezeigt, sonst geht der Bildschirm wieder zur letzten Benutzeroberfläche zurück. Bei Darstellung der Metadata für gewähltes Bild wird die in Metadata beschreibende Inhaltsegmente mit einen Polygon eingeschlossen, dadurch sind ihr mit Tastatur auswählbar und der Benutzer kann auch durch die Menüoption weitere Metadata für ausgewählte Segment abfragen, diese Funktion wird nicht in diesem Programm aufgebaut und nur in Menü als eine erweiterbare Funktion gezeigt.

In dem Aktivitätdiagramm (Abb.24a) gibt es insgesamt vier graphische Benutzeroberflächen, welche mit rotem Kreise markiert und jede davon ein View-Objekt des MVC-Paradigmas betrachtet werden kann. Aber gibt es keine weiteren Informationen über Model-Objekt und Controller-Objekt von diesem Diagramm lesen können, da in dem MVC-Paradigma nur die View-Objekte mit den Benutzer kommunizieren.

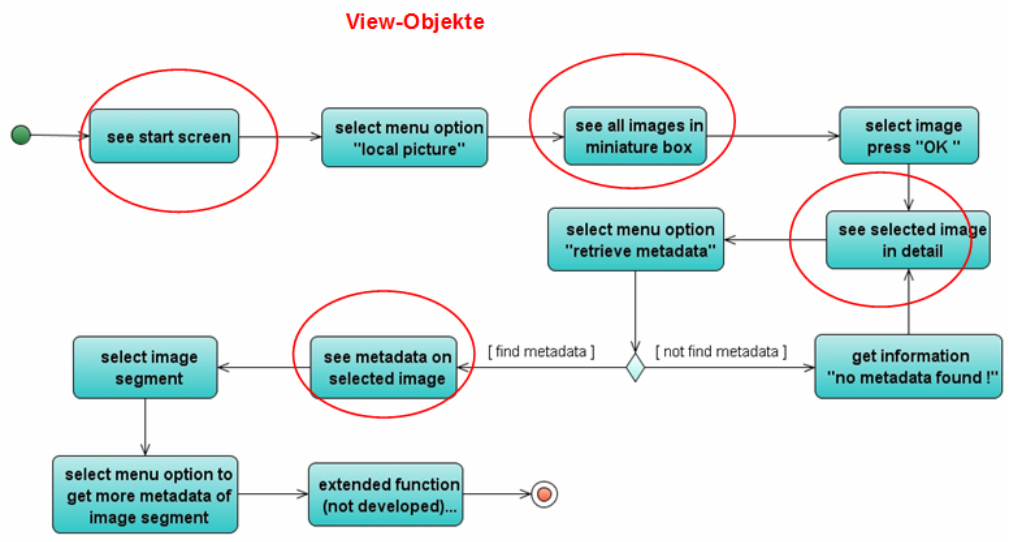

Abb.24a

Aktivitätsdiagramm von Benutzeransicht

Wird die Ansicht zur den Controller wechselt, wird die Model-Objekte mehr deutlich dargestellt. Die Abb.24b zeigt das Aktivitätsdiagramm von Controller-Ansicht. Im Vergleich mit Abb.24a hat das Diagramm die im Hintergrund liegende Aktivitäten des Programms gezeigt. Bei Aktivierung der Menuoption „local picture“ wird das Controller-Objekt eine Methode aufgerufen, die Bilddaten von dem Mobiltelefon suchen kann und in einem Model-Objekt konstruiert wird. Von der gefundene Bilder werden in einem Datenmodel geschrieben und dann in einer List zusammen gesammelt, die dann durch Controller-Objekt nach View-Objekt übertragt und gezeigt werden. In der gleichen weise wird die Metadata Retrieval auch durch einem Model-Objekt implementiert und danach erhaltende Metadaten in einer Datenstruktur speichert, welche auch als ein Model-Objekt aufgebaut wird.

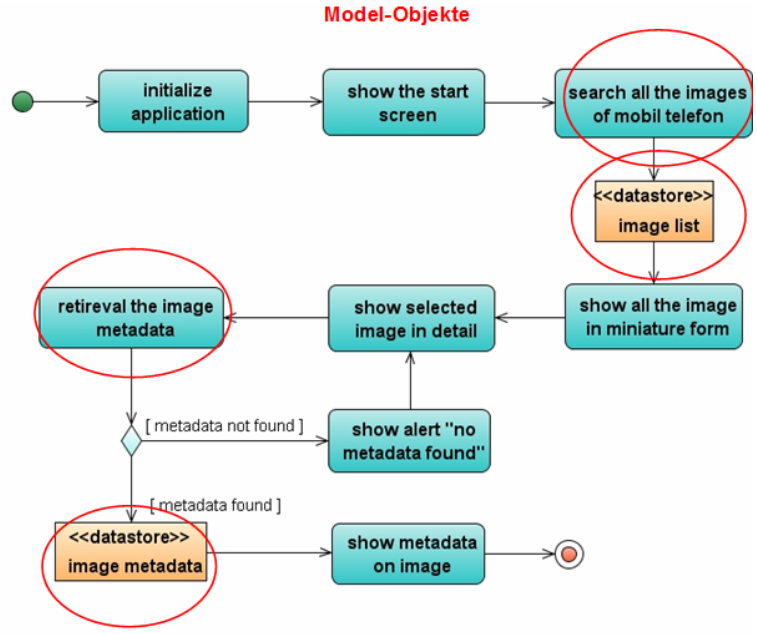

Abb. 24b

Aktivitätsdiagramm von Controller-Ansicht

6.2 Definition der Programmobjekte

Durch Analysis von dem Aktivitätsdiagramm im Abschnitt 6.1 werden folgende Programmobjekt in View-Objekte und Model-Objekte des MVC-Paradigmas zusammengefasst.

View-Objekte:

1. Benutzeroberfläche für Beginn des Programms(***Klasse StartScreen***);
2. Benutzeroberfläche für Auswahl der Bilddaten(***Klasse SelectImageScreen***);
3. Benutzeroberfläche für detailliierte Repräsentation des Bildes(***Klasse SchowImageScreen***);
4. Benutzeroberfläche für Repräsentation der Metadata des Bildes(***Klasse ShowMetadataScreen***);

Model-Objekte:

1. Model für Durchsuchung der Bilddaten, welche in einem Bestimmten Verzeichnis des Mobiltelefons gespeichert sind(***Klasse ImageSearchModel***);
2. Model für Erzeugung und Speicherung der von ImageSearchModel gefundenen Bilddaten(***Klasse ImageStoreModel***) ;
3. Model für Speicherung des einzelnen Bildes(***Klasse ImageBoxModel***)
4. Model für Retrieve des Bildmetadatas(***Klasse MetadataRetrievalModel***);
5. Model für Speicherung der von MetadataRetrievalModel zurückgeholten Metadaten(***Klasse ImageMetadatenModel***);
6. Model für Speicherung der Metadata des Bildsegments(***Klasse ImageSegmentModel***);

Die entsprechende Architektur zwischen den View-Objekt und Model-Objekte wird in Abb.25 dargestellt, welche das „Einzelner Controller MVC-Paradigma“ verwendet(sieh Abschnitt 3.7.1). Unten steht vier Model-Objekte, deren Inhalte bzw. Funktionen werden durch oben stehende vier View-Objekte dargestellt. Ein Controller-Objekt steht dazwischen, um die Funktionslogik zu interpretieren und Benutzeroberfläche zu wechseln.

Zwischen StartScreen und ImageSearchModel (dunkel blau Linie)

Nachdem MIDlet-Applikation startet, wechselt der Controller die Benutzeroberfläche in StartScreen, das die Funktion des ImageSearchModels in dessen Menü als eine Option „local picture“ abbildet.

Zwischen SelectimageScreen und ImageBoxModel (rot Linie)

Bei Aufruf des ImageSearchModels wird eine Durchsuchung der Bilder in Mobiltelefon implementiert, jedes gefundenes Bild wird im ImageModel-Datentyp gespeichert. Danach holt der Controller alle ImageModel-Objekte vom ImageSearchModel und übertragt ihnen zu SelectImageScreen zu zeigen.

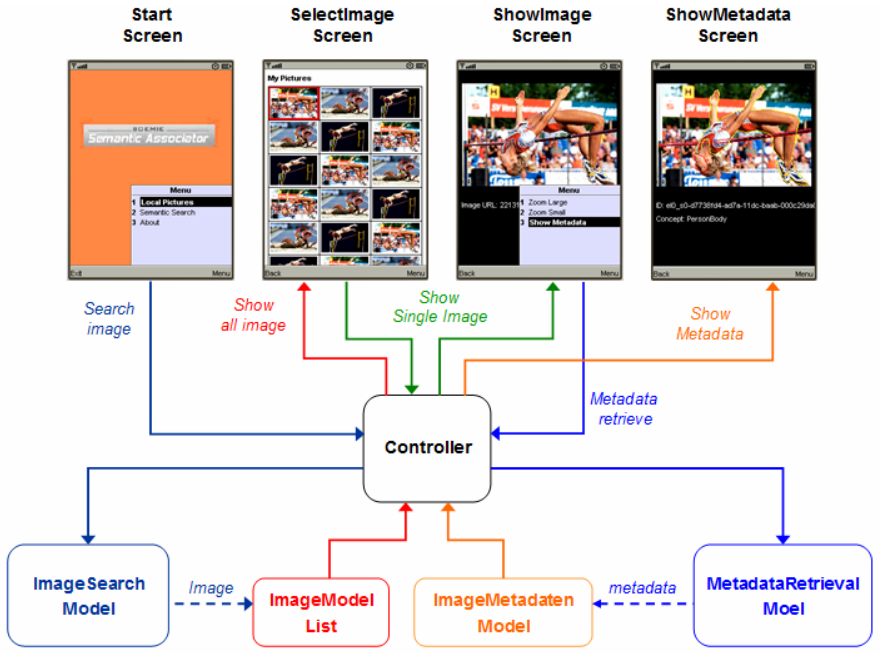

Abb.25
Programmarchitektur im MVC-Paradigma

Zwischen ShowImageScreen und MetadataRetrievalModel (grün+hell blau Linie)

Wird ein ausgewähltes Bild geklickt, wird das entsprechende ImageModel sofort von dem Controller nach dem ShowImageScreen übertragt, in dem wird dessen Bild vergrößert gezeigt.

Durch das Menü kann das Bild weiter bearbeitet werden und kann auch dessen Metadata von dem semantischen Web zurückholen. Diese Funktion wird im MetadataRetrievalModel realisiert und durch Menüoption „Show Metadata“ veröffentlicht.

Zwischen ShowMetadataScreen und ImageMetadataModel (orange Linie)

Wird die Menüoption „Metadata“ aktiviert, ruft der Controller gleich dem MetadataRetrievalModel auf, das Metadata des aktuell gezeigten Bildes zurückzuholen. Das von MetadataRetrievalModel erhaltende Metadata wird im ImageMdetadataModel-Datentyp gespeichert und dann von Controller im ShowMetadataScreen zugeteilt, graphisch darzustellen.

6.3 Entwurf der View-Objekte

In der Abschnitte 3.6 wird die Grundkenntnis für Anzeige des Mobiltelefons vorgestellt. Die Aufbau der graphischen Komponenten in dem Bildschirm des Mobiltelefon werden durch der Subklassen der Klasse *Screen* und der Klasse *Canvas* und sowie auch deren Subklassen in High-Level Anzeige bzw. In Low-Level Anzeige implementiert.

In dem Programm wird jedes im Metadata beschriebene Bildsegment mit einem roten Polygon markiert. Dazu steht keine entsprechende Methoden in den Subklassen der Klasse *Screen* für die Aufgabe zu Verfügung. Deswegen muss die Aufgabe mit der Klasse *Canvas* in Low-Level Anzeige implementiert werden. D.h. müssen alle vier Benutzerinterfaces die Klasse *Canvas* erben und die Methode *public void paint (Graphics g)* umschreiben. Außerdem der Datenverkehr zwischen View-Objekte(Benutzerinterface) und Model-Objekte(Benutzdaten) und die Anzeigewechsel von den Views sollen auch die MVC-Paradigma einhalten.

Das View-Objekt soll eine Menuliste anbieten, um die Benutzern das Programm benutzen anzuleiten und auch die Funktionen des Programms führen zu können. Der Benutzer kann durch Tastatur die Menuoptionen wählen und bestätigen oder abbrechen. Dazu muss das View-Objekt die Ereignis von den Benutzeraktionen abhören und dann nach Controller zu bearbeiten übertragen. Zur Aufbau der Menupunkte und Abhören von Ereignissen werden die Klassen *Command* und das Interface *CommandListener* eingeführt.

Eine Menuoption kann einfach mit Konstruktormethode der Klasse *Command* erstellt und deren Name, Typ und Priorität sind auch durch Parameter der Konstruktormethode definierbar. CommandListener ist ein Interface und deklariert nur eine Methode *commandAction(Command c, Displayable d)*, um die entsprechende Aktionen zu definieren, die werden später in dem View-Objekt umgeschrieben. Der folgende Code ist ein einfaches Beispiel des Benutzerinterfaces, das zwei Menuoptionen hat, deren ausgelöste Aktionen werden in der Methode *commandAction(Graphics g, Displayable d)* definiert.
Aber nach MVC-Paradigma sollen die Programmcode der Aktionen in Model geschrieben und von der Klasse *Controller* aufgerufen werden.

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Graphics;
import.javax.microedition.lcdui.Command;
import.javax.microedition.lcdui.CommandListener;

public class ExampleScreen extends Canvas implements CommandListener{
    private Controller controller;
    private Command menuoption1;
    private Command menuoption2;

    public  ExampleScreen ( Controller c ) {
        this.controller=c;
        this.addCommand ( getMenuOption1Command ( ) );
        this.addCommand( getMenuOption2Command ( ) );
            }

    protected void paint ( Graphics g ) { }
```

```
    public Command getMenuOption1Command() {
        menuOption1 = new Command ( "Option1", Command.ITEM, 1 );
        return menuOption1;
    }

    public Command getMenuOption2Command() {
        menuOption2 = new Command ( "Option2", Command.ITEM, 1 );
        return menuOption2;
    }

    public void commandAction ( Command c, Displayable d ) {
        if ( d == this ) {
            if ( c == menuOption1 ) {
                controller.methode1( );
            } else if ( c==menuOption2 ){
                controller.methode1( );
            }
        }

}
```

6.3.1 Klasse StartScreen

Die Klasse *StartScreen* ist als die erste Benutzerinterface in dem Programm definiert. Sie hat zwei Menuoptionen von *Exit* und *Local Pictures*, um das Programm zu beenden bzw. Lokale Bild-Ressourcen zu überblättern. Außerdem hat sie auch ein Hintergrundbild, deren Dimension soll nach realer Anzeigegröße anpassen, welche einfach mit der Methoden *getHeigh ( )* und *getWeight* ( ) von der Superklasse *Displayable* erhältlich ist und dann mit den Methode von der Klasse *ImageTools* die Bildsgröße ändern. Der folgende Code ist die Methode *paint ( )* in der Klasse *StartScreen*.

```
protected void paint (Graphics graphics) {

    graphics.setColor(255, 255, 255);
    graphics.fillRect(0, 0, getWidth(), getHeight());   // leeren das Bildschirm mit weißer Farbe

    backgroundImage= ImageTools.zoomImage ( getBackgroundImage(), getWidth(), getHeight(), false, false );
    graphics.drawImage( backgroundImage, 0, 0, Graphics.LEFT|Graphics.TOP );
}
```

Nach der Klick der Menuoption von *Local Picture* wird die Methode von *Controller* selectImage() aufgerufen, um Bilddaten zu überblättern. Die Code ist in der Methode *commandAction* wie folgend geschrieben.

```
if ( command == localPicturesCommand ) {
    controller.selectImage();
```

}

6.3.2 Klasse SelectImageScreen

Solange der Controller eine Bilddatenlist zuteilt, wird die Klasse *SelectImageScreen* alle Bilddaten von dieser Datenlist in der gleichmäßigen kleinen Box wie Abb.26 in Bildschirm des Mobiltelefons zeigen. Wo kommt aus die Bilddatenlist und was ist die Bilddaten weiß der Klasse *SelectImageScreen* nicht, sie macht nur die Aufgabe, die Bilddaten gemäß der Bilddatenlist visuell zu zeugen.

Abb.26
Benutzerinterface des Emulators von WKT2.5 für Auswahl der Bilder

Jede kleine gleichmäßige Imagebox wird von der Klasse *ImageBox* definiert. Der Instance der Klasse *ImageBox* ist ähnlich wie ein Behälter mit standarder Größe, um die verkleinerte Bilder zu füllen, außerdem definiert sie auch die Eigenschaften des Bildboxs wie Koordinate, beinhaltetes Bild, URI (Universal Ressource Identifier) des Bildes. Würde eine Bildbox ausgewählt, zeigt die Bildbox einen roten Rahm. Auswahl der Bilder wird durch Tastatur kontrolliert, die entsprechende Ereignisse werden in der Klasse *SelectImage-Screen* von der Methode *keypressed (int keyCode)* abgehört und bearbeitet.

```
public void keyPressed ( int keyCode ){
    switch ( getGameAction(keyCode ) ) {
      case RIGHT:
      right();
      break;
      case LEFT:
```

```
        left();
        break;
        case UP:
        up();
        break;
        case DOWN:
        down();
        break;
        case FIRE:
        fire();
        break;
      }
      repaint();
    }

    public void up ( ) { …}
    public void down ( ) {… }
    public void left ( ) { …}
    public void right ( ) { …}
    public void fire ( ) { …}
```

Würde eines Bildbox gewählt und Bestätigungstaste gedrückt, wird dieses Bildbox zu Controller übergegeben zur Vergrößerung.

```
public void fire(){
  controller.showImage(imageBox[currentRow][currentColumn]);
}
```

Bei der Controller der Auswahl des Bildes durch Tastatur muss eines Problem beachtet werden, dass gewilltes Bild vor dem Auswahl die untere oder obere Grenzlinie überschreitet, soll diese Bild nach der Auswahl nach oben bzw. unten verschieben.

6.3.3 Klasse ShowImageScreen

Diese Klasse hat nur die Aufgabe, das von Controller gegebenem Bild zu vergrößern. Damit kann der Benutzer die Details des Bildes sehen(Abb.27). Wozu wird wieder die Methode *zoomImage* von der Klasse *ImageTools* benutzet. Außerdem kann der Benutzer auch die Metadata des Bildes anfragen, solche Funktion wird in der Menuoption vorliegen. Solange diese Menuoption geklickt wird, wird diese Ereignis von der Klasse *KommandListener* gefangen (sieh unten Code) und entsprechende Methode von *Controller* aufgerufen, um die RacerPro-Server zu verbinden und die Metadaten des Bildes zu retrieve. Solche Aufgaben werden von dem Modell *MetadataRetriever* übergenommen, das wird in späterem Abschnitt weiter eingeführt.

```
public void commandAction(Command command, Displayable displayable) {
```

```
        if (displayable == this) {
          if (command == backCommand) {
             controller.switchDisplayable(null, controller.getSelectImageScreen());
          } else if (command ==zoomLargeCommand ) {

          } else if (command ==zoomSmallCommand ) {

          } else if(command==showMetadataCommand){
                controller.retrieveImageMetadata(imageURI);
          }
        }
    }
```

Die Metadata des Bildes in der Studienarbeit besteht aus vier Teilen: Image ID, Konzept und Profilpolygonen sowie die erweiterte Menü-Funktionen jeder Bildsegment. Diesen Metadaten werden mit OWL in einer Datei geschrieben und von RacerPro-System verwaltet. Die von der Antwort des RacerPro-System herausgezogene Metadaten müssen sich auch strukturiert werden. Zwei Datenstukturen von *ImageMetada* und *ImageSegment* werden zu Strukturierung von Metadaten des Bildes bzw. bestimmte Metadaten der Bildsegment beitragen. Die erstellte Instance der Klasse *ImageMetadata* wird in einer Hashtabelle gespeichert, um spätere wiederholte Prozess des gleiches Bilds zu vermeiden.

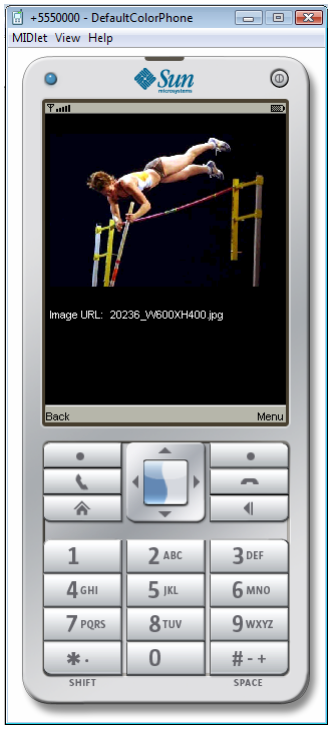

Abb.27
Benutzerinterface des Emulators von WKT2.5
für detaillierte Repräsentation der Bilder

6.3.4 Klasse ShowMetadataScreen

Nachdem die Antworten der Anfrage von Metadaten erhalten und geparst wurden, werden das Polygon jeder Bildsegment und das entsprechende Konzept durch *ShowMetadataScreen* gezeigt(Abb.28).

Jedes Bildsegment ist ein Objekt von der Klasse *ImageSegment*, welche die Attribute wie ID, Konzept, Polygon des Bildsegment-Objekts definiert. Durch die Richtungs-Tastatur können die Bildsegmente gewählt werden. Würde eine Bildsegment gewählt, wird seines Polygon gelb gewechselt und seines Konzept zugleich unten dem Bild gezeigt, sowie auch das Menü in den von dem Bildsegment zugehörige Funktionen gewechselt, wozu wird die Klasse *SegmentMenu* verwendet.

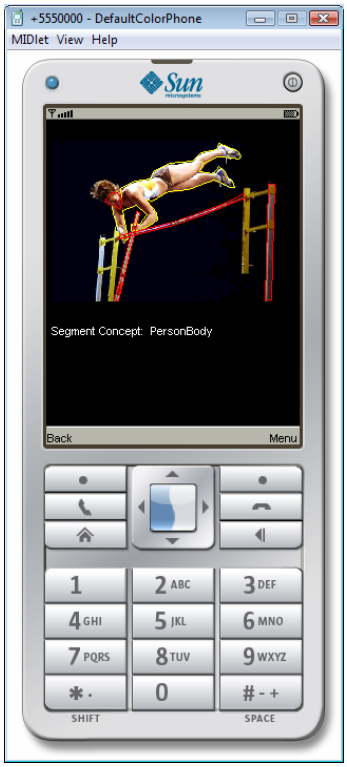

Abb.28
Benutzerinterface des Emulators von WKT2.5 für
Repräsentation des Polygons auf dem Bild

Zur Aufbau des Polygons des Bildsegments ist auch durch die Methode *graphic.drawLine (int x1, int y1, int x2, int y2)* von der Klasse *Graphics* realisiert, der Parameter ist zwei zweidimensionale Koordinate (x1,y1) und (x2,y2) für Anfangpunkt und Endpunkt der gerade Linie. Die Koordinaten von jedem Bildsegment sind eine eindimensionale Zahlenfolge in der Metadaten geschrieben und nach dem Retrieval und Parsen in dem Objekt der Klasse *ImageSegment* gespeichert. Hierbei wird auch eine Klasse *Coordinate* geschrieben um die eindimensionale Zahlenfolgen in zweidimensionale Koordinaten umzuwechseln. Bei Repräsentation des Polygons in dem *ShowMetadataScreen* werden jeweils zwei Koordinaten nacheinander herausgezogen und durch die *graphic.drawLine* Methode

verbunden. Es ist nennenswert, dass die letzte Koordinate mit erster Koordinate verbinden muss, um die geschlossene Polygon zu erzielen.

6.4 Entwurf der Model-Objekte

Jede abschließende Bearbeitung der Benutzerereignisse und Datenbearbeitung wird in Model geschaffen. In dieses Projekt umfasst das Model alle Operationsmethoden und Benutzdaten für Retrieve der Metadaten und auch die zusätzliche Anwendungen wie Durchblättern der Bilddaten von Mobiltelefon und Wechsel Menüzustand bei Auswahl unterschiedliche Bildsegmente. Hierbei werden drei Klassen definiert, um entsprechende Aufgaben zu durchführen.

6.4.1 Klasse MetadataRetrieverModel

Die Klasse *MetadataRetrieverModel* implementiert die zentrale Aufgabe in dem Programm, die Metadaten des Bildes abzufragen, welche aus folgenden drei Aktivitäten besteht.

1. Aufbau der Netzwerkverbindung mit dem entfernten RacerPro-Server;
2. Kommunikation mit RacerPro-System
3. Parsen der Response von RacerPro;

Die Netzwerkkommunikation mit RacerPro bezieht sich auf die Netzwerk-Programmierung in J2ME. J2ME bietet spezielle Klassen und Interfaces für verschiedene Kommunikations-Protokollen wie HTTP, FTP und Socket an.

6.4.1.1 Netzwerkverbindung mit RacerPro-Server

Die Interface ***Connection*** ist ein abstraktes Interface für alle Netzwerkverbindung, deren Subklassen werden für verschiedene Protokollen und Datentypen verwendet. Untere Abb.29 zeigt die hierarchische Relation aller Interfaces für Netzwerkverbindung in J2ME.

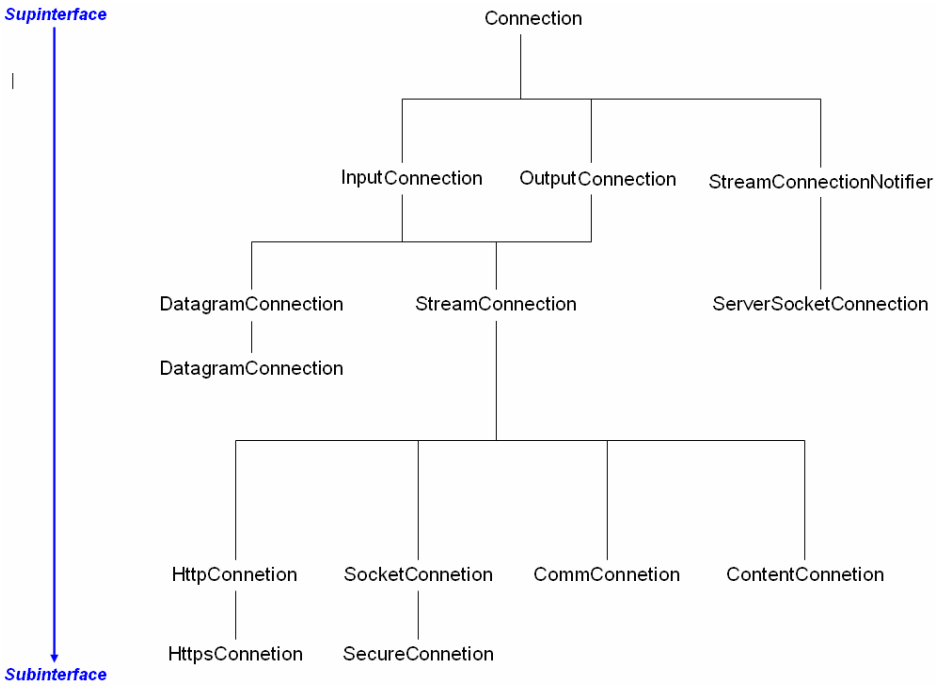

Abb.29
hierarchische Relation des Interfaces für Netzwerkverbindung in J2ME

In dem Projekt wird das Socket-Protokoll benutzt, um eine Verbindung mit dem entfernten Computer-Port aufzubauen. Jede Socket-Verbindung ist ein Objekt von der Interface ***SocketConnection***, welche ein Subinterface von ***StreamConnection*** ist. Eine Socket-Verbindung kann einfach mit der Methode *Connector.open ( )* erstellt werden, welche drei Parameter enthalten für *Kommunikationsprotokoll*, *Internetadresse* und *Prot*. Zum Beispiel eine Netzseite zu öffnen:

```
SocketConnection connection = (SocketConnection) Connector.open ( "socket://127.0.0.1:80" );
```

Der Rückgabewert von *Connector.open( )* in diesem Beispiel ist ein Objekt von der Interface *Connection* und wird mit dem SocketConnection-Datentyp gewechselt. Die Klasse *Connector* hat nur die Aufgabe, eine GCF(*Generic Connection Framework*)-Verbindung zu schalten und er verwaltet nicht die entsprechende Datenflussübertragung.

6.4.1.2 Kommunikation mit RacerPro-System

Die Verwaltung der Datenflussübertragung kann mit den in java.io.package definierte Input- und Output-Klassen realisiert werden, die für Schreiben der Daten in der öffneten Verbindung bzw. Lesen der Daten von der öffneten Verbindung zuständig sind. Die Klasse ***InputStream*** und ***OutputStream*** sind die Supklassen von allen Input- und Output-Klassen, welche für Übertragung unterschiedenen Datentypen zuständig sind. Eine InputStream oder OutputStream kann einfach mit der Methode

openInputStream( ) bzw. *openOutputSteam( )* von der Interface ***InputConnection*** bzw. ***OutputConnection*** erstellen.

```
InputStream input = connetion.openInputStream( );
OutputStream output = connection.openOutputStream( );
```

Solange ein MIDlet durch *startApp()* startet, wird eine Thread erstellt um das MIDlet auszuführen. Um das „Verkehrsstau“ im einzigen Thread des MIDlets zu vermeiden, wird die Socket-Verbindung in einem Runnable-Objekt konstruiert und in anderem Thread ausgeführt. Die Definition und Anwendung der Interface ***Runnable*** in J2ME ist sehr ähnlich wie in J2EE oder J2SE. Es gibt nur eine abstrakte Funktionsschnittstelle *run( )* in der Interface *Runnable* deklariert, die in dem Subklasse-Objekt weiter konstruiert wird. Bei Erstellung eines neues Thread-Objekt wird das Runnable-Objekt als einen Parameter benutzt und die *Run( )* Methode des Thread-Objekts wird von der Run( ) Methode der Runnabl-Objekte ersetzt. Folgend zeigt die Beispielcode,

```
public class MetadataRetrieverModel  implements Runnable{

        public void retrieveStart( ) {
              Thread thread=new Thread(this);
              thread.start( );
        }

       Public void run {
            Aufbau der Programmcode...
       }
}
```

Das Metadata Retieval wird durch Sendung der Ontologie-Anfragen nach RacerPro-System realisiert. Dieser Vorgang ist eine synchrone Kommunikation, nach Empfang der Antwort von RacerPro-System zieht das *MetadataRetrieverModel* mittels der Klasse ***StringTools*** die angeforderte Metadaten von der Antwort aus und speichert sie dann in lokaler Variable. Wie Abb.30 gezeigtes Sequenzdiagramm, bei Aktivierung der Menüoption „Metadata Retrieval“ von Benutzer ruft der Controller mit dem Parameter imageUrl die Funktionsschnittstelle des MetadataRetrieverModel-Objekts auf, um die entsprechende Metadaten zu haben. Die konkrete Arbeit zur Metadata Retrieval wird im MetadataRetrieverModel-Objekt erledigt. Nachdem alle Prozesse fertig gelaufen sind, holt der Controller das Ergebnis, ein ImageMetadata-Objekt, ab und übertragt es zum View-Objekt visuell darzustellen.

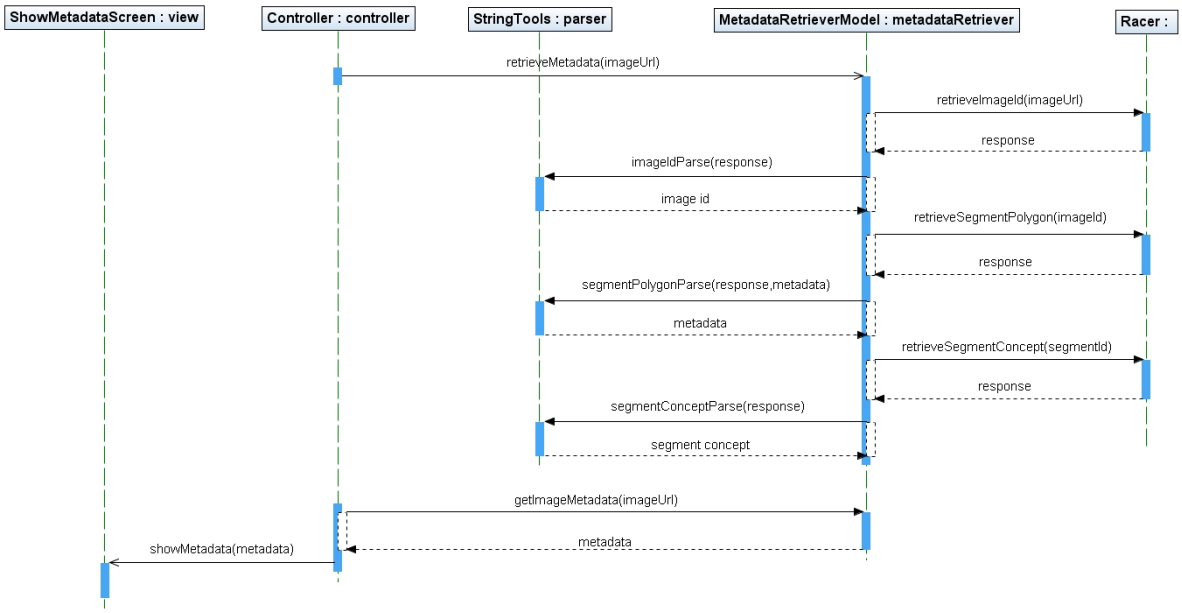

Abb. 30
Sequenzdiagramm

Die nach RacerPro gesandte Ontologie-Anfragen sind mit ***nRQL*** *(new RacerPro Query Language)* geschrieben, die eine Query-Sprache für Abfragen der Ontologie-Metadata von RacerPro-System verwendet wird. Zur Aufbau der Anfragen in der Studienarbeit soll eines Problem beachtet werden, dass URL mit das Zeichnung „ | “ geklammert werden muss, z.B.,

```
racerCommand = new String(
        "(retrieve (?x) (and (?x |http://www.boemie.org/BOEMIE_ontologies/mco.owl#Image|)" +
        "(?x (string= |http://www.boemie.org/BOEMIE_ontologies/mco.owl#hasURL| \""+var+"\"))))");
```

6.4.1.3 Parsen der Antwort von RacerPro

Die Klasse ***StringTools*** is als ein Werkzeug zur Parsen der Metadaten von RacerPro-Antworten benutzt. Es gibt verschiedene Methoden für entsprechende Metadaten zu parsen. Zur Abspeicherung der geparste Metadaten wird noch eine Datenstruktur benutzt. Diese Datenstruktur wird in der Klasse ***ImageMetadata*** definiert und jedes ImageMetadata-Objekt speichert alle Metadaten eines Bildes, wie Image ID, Segment-Polygon, Segment-Konzept.

Jedes Bild wird durch eine einzige Image ID identifiziert und enthält eine oder mehrere Segmenten, jede repräsentiert einen Gegenstand von diesem Bild und hat auch eine einzige ID, einen Konzeptname, und ein Polygon, welche durch viele zweidimensionale Koordinaten definiert und zur Darstellung der Figur des Segments verwendet wird. Zur Abspeicherung der Segmentes-Attributen wird die Hashtabelle benutzt. In J2ME ist Hashtabelle durch der Klasse ***Hashtable*** erstellt, welche

eine Indextstruktur wie die Klasse *Hashmap* in J2EE ist. Jedes in der Hashtabelle gelistetes Element ist ein Paar, der Schlüssel (key) und Hashwert (value) genannt. Jeder Schlüssel bildet einen einzigen Wert in der Hashtabelle. Zum Berechnen dieses Hashwertes wird ein Schlüssel benötigt, welcher dieses Objekt eindeutig identifiziert. Der Schlüssel und Hashwert können alle nicht-null Objekte zugewiesen werden, da die von der Klasse Objekt definiert werden. Hierbei wird Image ID als den Schlüssel betrachtet und Segment-Polygon und Segment-Concept werden zwei Hashwerten in zwei Hashtabellen betrachtet.

Die originale Antwort von RacerPro-System ist eine nicht verschlüsselte und direkt lesbare Zeichenkette, welche die angefragte Metadaten enthalten und nicht zu übersetzen und zu entschlüsseln benötigt. Außer Metadaten gibt es noch andere Zeichnen, Attributesnamen und Attributeswerten in der Zeichenkette, die für Benutzer unverständlich und sinnlos sind.

Mit der Methode *String.substring(int beginIndex, int endIndex)* können die gewünschten Teile, welche zwischen den Parametern befindet, von der Zeichenkette herausgezogen werden. Um die Position der zwei Parametern zu bestimmen wird die Methode *String.indexOf (String str, int fromIndex)* benutzt, sie geben einen Index zurück, bei dem die gegebener Zeichenkette „*str*“ nach der Beginnstelle „*fromIndex*“ erst mal auftritt, sonst -1 zurück. Folgendes Beispiel zeigt, wie wird die Zeichenkette von einer Klammer ausgeholt.

```
str="(Beispiel Zeichenkette)"

if((pos1=str.indexOf("(", 0))!=-1)
     if((pos2=str.indexOf(")", pos1))!=-1)
         substr=str.substring(pos1+1, pos2);
```

6.4.2 Klasse ImageMetadataModel

Das ImageMetadataModel ist tatsächlich eine Hash-Datenstruktur für Speicherung der Metadaten, welche von MetadataRetrievalModel aus dem semantischen Web zurückgeholt sind. Es hat zwei private Variable für die Attribute des Metadatas zu definieren:

***imageId**: Eine private Variable vom Typ String zu Speicherung der Image-ID des von dem Metadata geschriebenen Bildes.*
***segmentTable**: Eine private Variable vom Typ Hashtabell, welche die Segment-Objekte speichert, die den Inhalt des Bildes zusammen dargestellt, und eine Abbildung von Schlüssel(Segment-ID) auf Hashwert(Segments-Objekt) bietet.*

***Klasse ImageSegment**: eine Struktur zur Speicherung der Attributen des Segment-Objekt, wie Segment-ID, Segment-Konzept und Segment-Polygon und auch die entsprechende Methode zur Zugriff der Attributeswerte.*

Außerdem gibt es noch die entsprechende Methode zur Einfügung der Attributeswerte, welche durch MetadataRretrievalModel von Metadata ausgezogen sind, und sowie auch die Methode zur Zugriff der zwei private Daten.

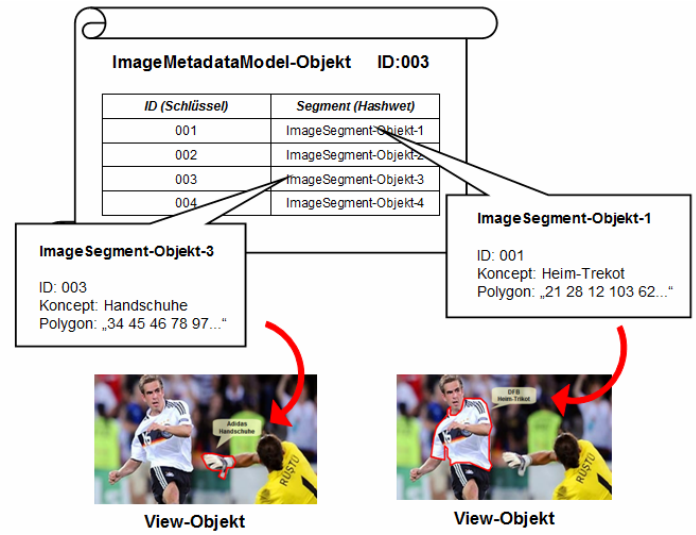

Abb.31
Hashtable im ImageMetadataModel-Objekt

Wie in Abb.31 gezeigt, jeder in der Hashtabelle gespeicherte Hashwert ist eines ImageSegment-Objekt, und enthalte eigene Attributeswerten, die als ein Model-Objekt des MVC-Paradigmas von View-Objekt visuell dargestellt werden können.

6.4.3 Klasse ImageModel

Die ImageBoxMode Klasse ist eine Struktur zur Speicherung des von ImageSearchModel im Mobiltelefon gefundene Bildes und sowie auch deren URL.

6.4.4 Klasse ImageSearchModel

Die Klasse *ImageSearchModel* bietet dem Controller eine Bilddatenlist, welche alle Bilderdaten, die im bestimmten Verzeichnis des Mobiltelefons gespeichert sind, in dem Typ ImageModel sammelt. Der Controller gibt diese Bilddatenlist als Parameter zum Objekt der Klasse *SelectImageScreen*, um die Bilddaten visuell abzubilden. Normalerweise soll das Datenbrowsen mit der Klasse *FileConnection* zu implementieren, um den Verzeichnispfad des Mobiltelefons bekommen und

verbinden. Aber da der Fokus dieses Projekts nicht in Bearbeitung des Files liegt, wird die Bilddatenlist direkt in dieser Klasse definiert.

6.5 Entwurf der Controller-Objekte

Das Stragegiemuster wird für Entwurf des Controller-Objekts verwendet, dessen Prinzip bereits im Abschnitt 2.7.2 vorgestellt. Die Schnittstellen aller Methoden, die in der Strategie benötigt werden, müssen zuerst in einer Abstrakten Klasse *Controlle*r deklariert werden. Der konkrete Code zur Implementierung der Strategie wird in verschiedenen Subklassen der abstrakten Klasse *Controller* konstruiert. In dieses Programm wird nur einzelnes Strategie bzw. nur ein Controller-Objekt verwendet für View-Objekte und Model-Objekte zu verwalten. Das „*Single Controller MVC-Paradigma*“ wurde im Abschnitt 3.7.1 diskutiert.

6.5.1 Klasse Controller

Die Klasse *Controller* wird mit Abstrakt-Typ deklariert, deren Methode werden gemäß der Aktivitätsdiagramm von Abb.17b definiert und müssen auch in Abstrakt-Typ deklariert werden.

```
abstract class Controller{
    // initialisieren aller Anwendungsdaten
    abstract public void screenInitialize( );
    // wechselt den Bildschirm in der Benutzeroberfläche, welche den Startzustand des Programms  zeigen kann
    abstract public void showStartScreen( );
     // holt alle Bilder von Mobiltelefon oder bestimmte Verzeichniss des Mobiltelefon, welche in einem Vector-Objekt gespeichert werden
    abstract public Vector getImageList( );
     // wechselt den Bildschirm in der Benutzeroberfläche, welche alle in Parameter ImageList gespeicherte Bilder im miniraturen Form zeigten
      kann
    abstract public void showImageList( );
    // wechselt den Bildschirm in der Benutzeroberfäche, welche ein einzelnes Bild in detailiert
    abstract public void showSingleImage( );
   // retrieve die Metadata von dem Bild, welche mit gegebenem Parameter imageUrl identifiziert wird
    abstract public void retrieveMetadata( String imageUrl);
  // wechselt den Bildschirm in der Benutzeroberfläche, welche den gegebene Parameter imageMetadata im Bildschirm darstellen kann.
    abstract public void showImageMetadata(String imageUrl);
  // zur Wechsel der Benutzeroberfläche
    abstract public void switchDisplayable (Alert alert, Displayable nextDisplayable);
}
```

Wie oberer Beispielcode gezeigt, dass jede in Controller deklarierte abstrakte Methode eine Aktivität von dem Programmlauf steuert, die konkrete Code und Funktionslogike werden bei der Subklasse der Klasse *Controller* weiter konstruiert. Das Anwendungziel jeder Methode und sowie die Parametern werden als Kommentar in grauen Texte geschrieben.

6.5.2 Klasse MMR-Controller

Die MMR-Controller ist eine Subklasse von der abstrakten Klasse *Controller* und konstruiert er alle in dem Controller deklarierte abstrakte Methoden mit konkretem Programmcode. Wie bei folgendem Beispielcode gezeigt, dass alle View-Objekte und Model-Objekte als privaten Variablen in ihrem Controller-Objekt deklariert werden müssen, damit die Controller-Objekte deren Methode bzw. Daten verwaltet.

```
Public class MMR-Controller extends Controller{
   private MainMidlet mainMidlet;
   private StartScreen startScreen;
   private SelectImageScreen selectImageScreen;
   private ShowImageScreen  showImageScreen;
   private ShowMetadataScreen showMetadataScreen;
   private AlertScreen alert;
   private MetadataRetrieveModel metadataRetriever;
   private ImageSearchModel imageCollector;

   public void Controller ( Midlet midlet ){ ...}
   public void screenInitialize( ) { …}
   public void showStartScreen( ) { …}
   public Vector getImageList( ){ …}
   public void showImageList( ){ … }
   public void showSingleImage( ) { …}
   public void retrieveMetadata( String imageUrl ) { … }
   public void showImageMetadata(String imageUrl ){ …}
   public void switchDisplayable ( Alert alert, Displayable nextDisplayable ){ … }
   …

}
```

7 Zusammenfassung

In dieser Studienarbeit wurde die Anwendung des MVC-Paradigmas zur Unterstützung des Multimedia-Retrievals im Mobiltelefon untersucht. Das MVC-Paradigmas wird als ein Entwurfsmuster umfangreich in unterschiedlichen Programiersprachen für objektorientierten Programm-entwürfe verwendet. Das Prinzip ist Trennung aller Daten und Methoden des Programms in drei Komponente: Model, View und Controller. Alle in dem Programm ausgestellten und bearbeitenden Daten werden in Model-Objekte konstruiert und durch View-Objekte in der Ausgabe-einrichtung dargestellt. Dazwischen benötigen Steuerungen und Funktionslogik wird in Controller-Objekt definiert. Durch solche Strukturteilung lässt sich die jede Komponente die eigene Aufgabe unabhängig von der anderen Komponente schaffen können. Damit die Wiederverwendung und Austausch einzelner Komponente ermöglich und jede Komponente auch durch verschiedenen auf der Komponente spezialisierte Entwickler entwickelt werden kann. Dadurch erhöht die Arbeitsleistung und spart die Zeit und Kosten.

Die Vorteile des MVC-Paradigmas sind in den großen, komplexen und auf der graphischen Benutzeroberfläche basierten objektorientierte Programmentwicklung sehr offensichtlich, (aber für den gleicherweise auf der graphischen Benutzeroberfläche basierten J2ME-Programmentwicklung ist die Anwendung des MVC-Paradigmas nicht so viel.) aber solche Trennung der Programmcode in drei Komponente führ zu einem großen Aufwand von zusätzlicher Kommunikationen und Datenverkehren zwischen jeder Komponente. Damit die Kapazität und Komplexität der Programmcode erhöht werden und die Programmleistung und Ausführungsgeschwindigkeit reduziert. Dieser Nachteil ist für den kleine und Hardware beschränkte J2ME-Programme nicht akzeptierbar. Mit schneller Entwicklung der eingebetteten Technik und Hardware-Technik ist die Ausführung der großen und komplexen Programme in kleinem Mobiltelefon ermöglicht. Wie das MVC-Paradigma in praktischer J2ME-Programmentwicklung verwendet wird, wurde als einen Schwerpunkt in dieser Studienarbeit untersucht.

Der andere Schwerpunkt von der Studienarbeit liegt, wie das Programm für Multimedia Retrieval in J2ME konstruiert wird. Bezüglich J2ME-Programmierung bin ich noch ein Anfänger. Deswegen wurden viele Grundkenntnisse über J2ME sowie GUI-Programmierung, Netzwerkprogrammierung und Entwicklungsumgebung in der Studienarbeit erläutert. Dann wurde auch die Anwendungs-möglichkeit der häufig benutzten Architekturen des MVC-Paradigmas in J2ME-Programmierung diskutiert. Danach wurden die Motivation und Arbeitprinzip von Multimedia Retrieval im Mobiltelefon kurz vorgestellt.

Nach eingehende Untersuchung und Verstehen von dem MVC-Paradigma und der J2ME-Programmierung beginnt der praktische Programmentwurd. In der Teil wurde sich mit damit

auseinandergesetzt, wie die Programmstruktur gemäß des MVC-Paradigmas in drei Teilen geteilt wird, Welche Rolle des MVC-Paradigmas die führende Klassen gespielt werden und wie die Klassen nach MVC-Paradigma gegeneinander funktioniert. Am Ende wurden die Beispielcode zur Konstruktion der führenden Klassen ausführlich vorgestellt. Durch diese Studienarbeit habe ich mehr Wissen und praktische Erfahrungen über das MVC-Paradigma und Entwurfsmustern erworben, es ist für spätere objektorietierte Programmentwürfe und Programmentwicklung sehr hilfreich.

Reference

[IoSW] Introduce of the semantic web from W3C, URL: http://www.w3.org/2001/sw/

[PL] Alexander, C., Ishikawa, S., Silverstein, M., *A Pattern Language*, New York: Oxford University Press, 1977

[EV] Alan Shalloway, James R.Trott, *Entwurfsmuster verstehen*, Quotation from page 86, 2003

[DP] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*,1996

[TR] Trygve M. H. Reenskaug, *Technical note from Xerox Parc*, 1979
URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[GE] Glenn E. Krasner, Stephen T. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, SIGS Publications, Denville, NJ, USA, 1988

[SB] Steve Burbeck, *Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC)*, 1992, URL: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[SMj] Artikel von Sun Microsystem, *Introduction to the Java ME Platform*,
URL: http://java.sun.com/javame/technology/index.jsp

[SMc] Webpage von Sun-Microsystem für Connected Limited Device Configuration (CLDC),
URL: http://java.sun.com/products/cldc/

[NF6] S60-Plattform Download in Nokia Forum,
URL: http://www.forum.nokia.com/main/resources/tools_and_sdks/index.html#overview

[NF] Nokia Forum, url: http://www.forum.nokia.com

[NFo] Document about OTA from Forum Nokia: *Settings for OTA Download of MIDlets v1.0*
URL: http://www.forum.nokia.com/info/sw.nokia.com/id/685dddec-38c8-419d-aaca-3639b8f03705/COD_intro.pdf.html

[NFc] Document about OTA from Forum Nokia: *OTA download for generic content-Introduction*
URL: http://www.forum.nokia.com/info/sw.nokia.com/id/685dddec-38c8-419d-aaca-

3639b8f03705/COD_intro.pdf.html