Katrin Hänsel

Modernisierungskonzept der Architektur von iBEAM

eingereicht als

Bachelorarbeit

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fachbereich

Mathematik Naturwissenschaften Informatik

Mittweida, 2010

Erstprüfer: Prof. Dr.-Ing. W. Schubert

Zweitprüfer: Dr.-Ing. B. Sünder

Vorgelegte Arbeit wurde verteidigt am:

Bibliographische Beschreibung

Hänsel, Katrin:

Modernisierungskonzept der Architektur von iBEAM,

2010 – Mittweida, Hochschule Mittweida, Fachbereich Mathematik Naturwissenschaften Informatik, Bachelorarbeit

Kurzreferat

Die vorliegende Bachelorarbeit beschreibt Konzepte zur Architekturmodernisierung der Cocoa-Touch-basierten Applikation „iBEAM“ für mobile Apple Produkte. Diese „App“ dient zur Darstellung serverseitiger Messdaten. Dabei werden Konzepte zur dynamischen Gestaltung der Netzwerkschicht untersucht und auf Möglichkeiten der Touch-gesteuerten Oberflächenbedienung eingegangen.

Stichwörter: Cocoa Touch Framework, Netzwerk-Protokoll, Natural User Interface, Oberflächenkonzepte iPhone

Abstract

This thesis deals with concepts for the architectural modernization of the cocoa touch based application “iBEAM” for Apple’s mobile devices. This “App” provides visualizations for server based measuring data. Therefore, concepts for the dynamic design of the networking interface will be analyzed and it will be dwelled on possibilities of a touch operated user interface.

Keywords: Cocoa Touch Framework, network protocol, natural user interfaces, user interface concepts for iPhone

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Glossar und Abkürzungen

App:	Kurzform für Application – Bezeichnung für Anwendungen im Smartphone-Bereich
Apple AppKit:	*Apple Applikation Kit:* Apples Framework für die Erstellung von graphischen Oberflächen.
Autorelease Pool:	Ein Werkzeug zur manuellen Speicherverwaltung ohne Garbage Collection unter Cocoa.
CEA:	*Components for Evaluation and Analysis:* Standard zur Entwicklung von komponentenbasierten Applikationen zur Auswertung und Analyse von Messdaten.
CLI:	*Command Line Interface:* Kommandozeileninterface zur Interaktion mit einem Computer. Die Informationen und Anweisungen werden in Textform angezeigt und aufgenommen.
CLI:	*Command Line Interface:* Kommandozeile (z.B. MS-DOS)
Cocoa-Framework:	Eine Sammlung von Basisklassen und Programmierwerkzeugen zur Programmierung von MacOS und iOS Anwendungen. Es basiert auf der Programmiersprache Objective-C.
Endianness (Byte-Reihenfolge):	Reihenfolge der Bytes einer Zahl im Speicher. Big Endian: Byte mit dem höchstwertigen Bit wird zuerst (an der kleinsten Speicheradresse) gespeichert. Big Endian: Byte mit dem niedrigwertigsten Bit wird zuerst (an der kleinsten Speicheradresse) gespeichert.
Enterprise-Datentyp:	Datentypen, welche von EnCom Protokoll unterstützt werden. Übersicht siehe 0.
Garbage Collection:	Automatische Verwaltung des Speichers eines Programmes
GUI:	*Graphical User Interface:* Graphische Benutzeroberfläche.
GUI:	*Graphical User Interface:* Graphische Benutzeroberfläche (z.B. Windows Oberfläche)
Header:	Bezeichnet im EnCom-Protokoll die Kopfdaten, die mit einer Nachricht übers Netzwerk übertragen werden. Beinhaltet allgemeine Informationen zur Nachricht, wie Typ, Kompression, ID und Größe der eigentlichen Daten (Payload).
IEEE:	*Institute of Electrical and Electronics Engineers*
iPhone OS/iOS:	Das Betriebssystem für Multi-touch Geräte wie iPhone, iPad und iPod. Ab Juni 2010 mit iOS bezeichnet.
(Enterprise) Item	Ein (Enterprise) Item stellt ein Objekt zur Kapselung von Daten dar und besitzt zur Einordnung eine eindeutige ID.

	Nummerische Items können außerdem statistische Informationen enthalten.
KVO	*Key-Value Observing:* Kommunikationsservice, bei dem Änderungen an einer Objekteigenschaft verbreitet werden.
Mac OS X:	Betriebssystem von Apple für die hauseigenen Macintosh-Computer.
Map (Mappe):	Datenstruktur, welche eine Schlüssel-Wert-Zuweisung darstellt.
Memory Management:	Speicherverwaltung
NUI:	*Natural User Interface:* Natürliche Benutzeroberfläche.
NUI:	*Natural User Interface:* natürliche Benutzeroberflächen (z.B. Multi-Touch-Oberflächen, Wii Spielekonsole)
Objective-C:	Objektorientierte Programmiersprace, die mit C verwandt ist und deren Syntax an die Sprache Smalltalk erinnert.
OS:	*Operating System:* Betriebssystem
Payload:	Bezeichnet im EnCom-Protokoll die reinen Nutzdaten, die mit einer Nachricht über das Netzwerk übertragen werden. Beinhaltet die eigentliche Nachricht und folgt auf den Header.
Protokoll:	Ein Protokoll bezeichnet in der Informatik eine Sammlung von Regeln zur Datenübertragung zwischen zwei Kommunikations-partnern.
Serialisierung:	Vorgang der Umwandlung eines Objektes in eine transportierbare oder speicherbare Form. z.B. XML-Serialisierung, binäre Serialisierung
Smartphone:	Leistungsfähiges Mobiltelefon, welches um die Funktionalitäten eines Personal Digital Assistent (PDA) erweitert wurde
TCP:	*Transmission Control Protocol:* Verbindungsorientiertes, zuverlässiges Netzwerkprotokoll
UMTS:	*Universal Mobile Telecommunications System:* Standard zur Datenübertragung im Mobilfunk
WLAN:	*Wireless Local Area Network:* Drahtlose Netzwerkverbindung
Xcode:	Entwicklungsumgebung

Vorwort

Diese Arbeit richtet sich vorrangig an Personen, welche im Bereich der Informatik und im Speziellen der Anwendungsentwicklung und –programmierung tätig sind und ein gewisses Grundwissen im Bereich der Softwareentwicklung und der Programmierung mitbringen.

Des Weiteren ist zu beachten, dass wenn im Folgenden von iPhone gesprochen wird, auch meistens der iPod Touch und das iPad gemeint sind.

Danksagung

An dieser Stelle möchte ich mich für die Unterstützung und Geduld während der Umsetzung dieser Arbeit bedanken. Mein Dank geht dabei an Prof. Wilfried Schubert für die Unterstützung und Bereitschaft, immer ein wenig Zeit für mich zu finden. Außerdem möchte ich mich bei Dr. Bernhard Sünder für die Möglichkeit und Mittel, mich in dieses schon länger vorhandene Interessengebiet einzuarbeiten bedanken. Des Weiteren richte ich meinen Dank an Markus Ritter für seine endlose Geduld und aufmunternden Worte.

1 Einleitung

1.1 Die Firma AMS

Die Firma „Gesellschaft für angewandte Mess- und Systemtechnik“ (AMS) wurde im Jahre 1993 gegründet und übernahm den Vertrieb der Messtechniksoftware BEAM von der Firma Hottinger Baldwin Messtechnik, welches als das weltweit erste Programm mit einer graphischen Fenstertechnik gilt und zur Datenerfassung und –auswertung auf den Markt gebracht wurde. Dafür erhielt die Firma AMS im Jahre 1995 den Innovationspreis des Freistaates Sachsen. Ein Jahr darauf wurde die bisher nur Macintosh Nutzern zugängliche Software auf Windows 95 portiert.

Im Jahre 1999 begann die AMS GmbH mit der Entwicklung einer plattformunabhängigen Auswertesoftware, welche sich auf Java stützte. Diese sollte vor allem leistungsstark und in vielen Sprachen verfügbar sein, um somit international einsetzbar sein zu können. Das neue Produkt jBEAM sollte auch sehr auf die Verteilung im Netzwerk spezialisiert sein.

Im Laufe der Jahre setzen drei der größten, deutschen Automobilhersteller und der weltweit führende Anbieter im Bereich der Antriebstechnik jBEAM als Datenauswertungssoftware ein.

Doch die Firma AMS verfolgt auch immer wieder innovative Ziele. So wurde im Jahre 2009 mit der Programmierung einer prototypischen, mobilen Visualisierungssoftware für die mobilen Multi-Touch Geräte aus dem Hause Apple, begonnen. iBEAM soll vor allem mit den Produkten BEAM und jBEAM interagieren, um somit Messungen und Berechnungen dieser darstellen und auch steuern zu können. Die Zukunftsvision reicht soweit, dass der Nutzer somit die Möglichkeit bekommt, entfernte Prozesse von seinem iPhone, iPod Touch oder iPad zu steuern und zu überwachen. [AMS10]

1.2 Der Aufstieg der Firma Apple

Am 9. Januar 2007 präsentierte Apples CEO Steve Jobs, das iPhone. Dies ist das erste Mobiltelefon aus dem Hause Apple.([Stäuble09] , S.1) Es besitzt keine herkömmliche Tastatur, sondern einen Touch-Bildschirm und ermöglicht somit revolutionäre Bedienkonzepte. Das iPhone kann zwar nicht als erstes Smartphone mit dieser Technologie angesehen werden, jedoch gilt es als Vorreiter auf diesem Gebiet. Auch war es mit dem iPhone zum ersten Mal möglich in einer außerordentlich guten Qualität Web-Inhalte und E-Mail-Funktionen zu nutzen.

Wenige Monate nach der Veröffentlichung des iPhones brachte Apple auch einen iPod mit der neuen Touch-Technologie auf den Markt – den iPod Touch. Dieser umfasst eine ähnliche Funktionalität wie das iPhone, jedoch ist es mit ihm nicht möglich zu telefonieren.

Auch kann nur per WLAN auf Internetinhalte zugegriffen werden und nicht wie beim iPhone über UMTS.

Im Jahre 2010 stellte Apple sein neuestes Produkt im Bereich der mobilen, berührungsgesteuerten Geräte vor. Das iPad ist größer als ein iPod Touch oder iPhone und bietet somit auch ein wesentlich größeres Display, welches neue Möglichkeiten für Anwendungen eröffnet. Auch auf dem iPad läuft das Apple Betriebssystem *iOS*.

Anfänglich verweigerte Apple aus Sicherheitsgründen den Entwicklern die Programmierung eigener Applikationen für das iPhone. Nach Protesten lockerte Apple diese Konvention jedoch und veröffentlichte ein SDK, mit dem es dem Entwickler möglich wurde, eigene Apps zu programmieren und in dem dafür geschaffenen App-Store anzubieten. Dieser bietet die einzige Möglichkeit, Apps auf legalem Weg zu verbreiten. Jedoch behält sich Apple das Recht vor, jede Applikation selbst zu bewerten und unter Umständen die Aufnahme in den App-Store zu verweigern.

1.3 Problem- und Zielstellung

Seit der Entwicklung des App-Store und damit der Möglichkeit, seine eigens für das iPhone entwickelten Applikationen in diesem anzubieten, nutzen viele diesen Weg, um mit raffinierten Anwendungen auf sich aufmerksam zu machen. Dabei ist die Auswahl groß – von Spielen über Gadgets für den Alltag bis hin zu professionellen Anwendungen ist alles vertreten. Viele Firmen bieten schon Apps an, um somit Kunden den mobilen Zugang auf ihre Dienste zu gewähren. Oft auch aus dem Grund, dass es nicht möglich ist, auf dem iPhone Flash Inhalte wiederzugeben.

Auch die iPhone Applikation iBEAM arbeitet als Visualisierungsclient für messtechnische Daten und bietet dem Nutzer einen mobilen Zugriff auf Daten und Dienste eines kompatiblen Servers und visualisiert serverseitiger Messdaten und Berechnungen. Da das Produkt schon als Prototyp vorliegt, besteht die Hauptaufgabe in der Umstrukturierung, Stabilisierung und Erweiterung. Hierbei ist das Ziel wiederum auch nur ein Prototyp.

1.4 Gliederung der Arbeit

Diese Arbeit wird sich dazu zuerst mit der Evaluierung der bisherigen Funktionalitäten und Voraussetzungen des Produktes iBEAM befassen, um dem Leser einen besseren Einstieg in Nachfolgende Kapitel zu gewähren. Hierzu wird zuerst das Produkt an sich und danach die Kerntechnologie namens EnCom evaluiert. In dem nachfolgenden Kapitel werden Konzepte und Argumente für Strukturerneuerungen des iBEAM Kerns vorgestellt und auch der Umfang dieses Projektes abgesteckt. Nachfolgend soll ein Einblick in die Programmiersprache Objective-C und das Cocoa Framework gegeben werden, um ein Grundverständnis für die iPhone-Programmierung zur Verfügung stehenden Technologien zu geben. Dies soll eine Grundlage für die Implementierungsdetails geben,

welche folgen. Einige Performance-Tests sollen den Nutzen der Verbesserungen an iBEAM aufzeigen. Zu guter Letzt werden Konzepte zur Oberflächengestaltung untersucht. Hierbei soll auch auf die Gestaltung einer Oberfläche für das iPad eingegangen werden.

1.5 Abgrenzungen

Ziel dieser Arbeit ist die Erneuerung der prototypischen Applikation iBEAM. Dies umfasst vor allem die Erneuerung der Architektur des Kerns um somit eine stabile Grundlage für die aufsetzende Applikation zu schaffen. Aber auch die graphische Oberfläche soll erneuert werden und auf das neue Gerät iPad portiert werden. Hierbei ist im Auge zu behalten, dass das Endprodukt wieder nur einem Prototyp entsprechen wird und zukünftig einer Weiterentwicklung bedarf.

2 Das Produkt iBEAM

Das Produkt iBEAM ist eine Cocoa-Touch-Applikation für mobile Geräte der Firma Apple wie iPhone und iPod. Es wurde im Jahre 2009 von der AMS Gesellschaft für angewandte Mess- und Systemtechnik mbH entwickelt und steht eng in Verbindung mit deren anderen Softwarelösungen BEAM und jBEAM. Diese werden zu Beginn dieses Kapitels näher beleuchtet und danach wird auf die bisherigen Funktionalitäten von iBEAM eingegangen.

2.1 Die korrespondierenden Produkte zu iBEAM

iBEAM korrespondiert vor allem mit den Produkten jBEAM und BEAM. Es ist jedoch auch möglich, iBEAM mit anderen Programmen, welche das EnCom Protokoll nutzen, interagieren zu lassen. Hierzu sei zu erwähnen, dass sich die Beispiele dieser Arbeit jedoch immer auf die Kommunikation von iBEAM mit jBEAM als Serverapplikation beziehen.

2.1.1 jBEAM

jBEAM ist das Hauptprodukt der Firma AMS. Es ist ein Java-basiertes Auswertesystem mit einem breiten Einsatzspektrum. Es ist auf die Visualisierung von Messdaten spezialisiert und ist durch die Java Technologie plattformunabhängig einsetzbar. (Weitere Informationen: Webseite [AMS10] , Komponentenbeschreibung: [jBEAM10])

jBEAM orientiert sich am ASAM-CEA Standard. Dies bedeutet, dass es komponentenbasiert aufgebaut ist und durch Bereitstellung von Schnittstellen, dem Anwender eine freie Erweiterbarkeit mittels eigens entworfener CEA-Komponenten bereitstellt.

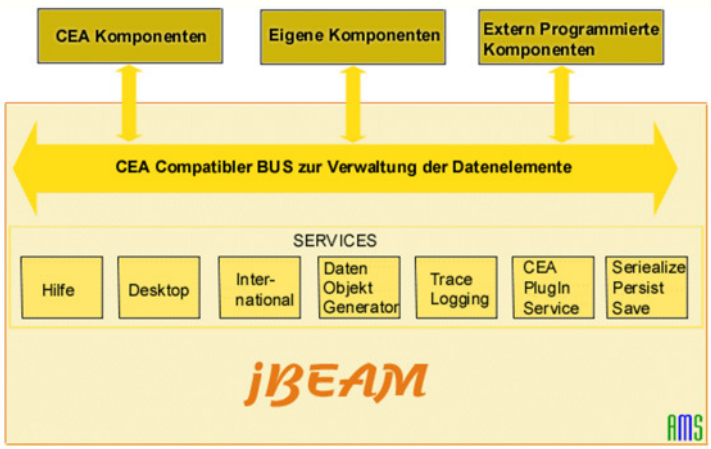


Abb. 2-1: Schema des CEA-Konzepts (Quelle: [AMS10])

Die Grundarbeitsweise von jBEAM basiert darauf, dass Messdaten als Kanäle auf dem CEA-Bus abgelegt werden und Visualisierungs- bzw. Berechnungskomponenten auf diese Daten zugreifen können. Dabei arbeitet jBEAM als Full-State-Application. Dies bedeutet,

dass Berechnungs- und Visualisierungskomponenten sich bei den Kanälen anmelden und somit über Änderungen der Daten informiert werden und ihre Berechnungen oder Anzeigen sofort aktualisieren können.

JBEAM kann auch als verteilte Server- oder Client-Applikation eingesetzt werden. Dies geschieht mit dem eigens im Hause AMS entwickelten EnCom Protokoll. Somit ist es möglich, Berechnungen und Daten auf mehrere jBEAM Server zu verteilen und von EnCom Clientapplikationen visualisieren zu lassen.

2.1.2 BEAM

BEAM ist die Standardauswertesoftware für Windows und dient der Datenerfassung und Auswertung vor Ort.

In Gegensatz zu jBEAM ist BEAM mehr auf die Messung als auf die Visualisierung und Auswertung spezialisiert. Es unterstützt eine Vielzahl von Messgeräten verschiedenster Firmen und ist leicht zu bedienen.

Auch BEAM unterstützt das EnCom Protokoll und kann als EnCom-Server genutzt werden.

2.2 Funktionalität von iBEAM aus Anwendersicht

iBEAM ist das jüngste Kind der AMS GmbH. Es ist für die Apple Produktreihe iPhone, iPod Touch und iPad konzipiert und ermöglicht die Visualisierung von Mess- oder Prozessdaten für unterwegs.

Dabei werden mittels des im Hause AMS entwickelten EnCom Protokolls, Daten zwischen iBEAM und einer EnCom Serverapplikation (z.B. jBEAM oder BEAM) übertragen. Nach einer erfolgreichen Verbindung zum Server, werden alle unterstützten Datenobjekte[1] des Servers angezeigt und können abonniert werden, so dass iBEAM über jede Änderung dieser Datenobjekte informiert wird und in Echtzeit auf dem neusten Stand bleibt. Die Daten der abonnierten Objekte können als Wertereihen angezeigt und graphisch visualisiert werden. Dabei ist zurzeit nur die Visualisierung einer eindimensionalen Datenreihe mittels eines Liniengraphen möglich.

Im Folgenden soll anhand eines Beispiels der Funktionsumfang erläutert werden. Hierzu wurden innerhalb von jBEAM, welches in diesem Beispiel als Server-Applikation dient, drei Kanäle mit Gleitkommawerten angelegt. Die Darstellung in jBEAM sieht wie folgt aus:

[1] Es werden nur Double-, Float- oder Integerwerte bzw -kanäle unterstützt

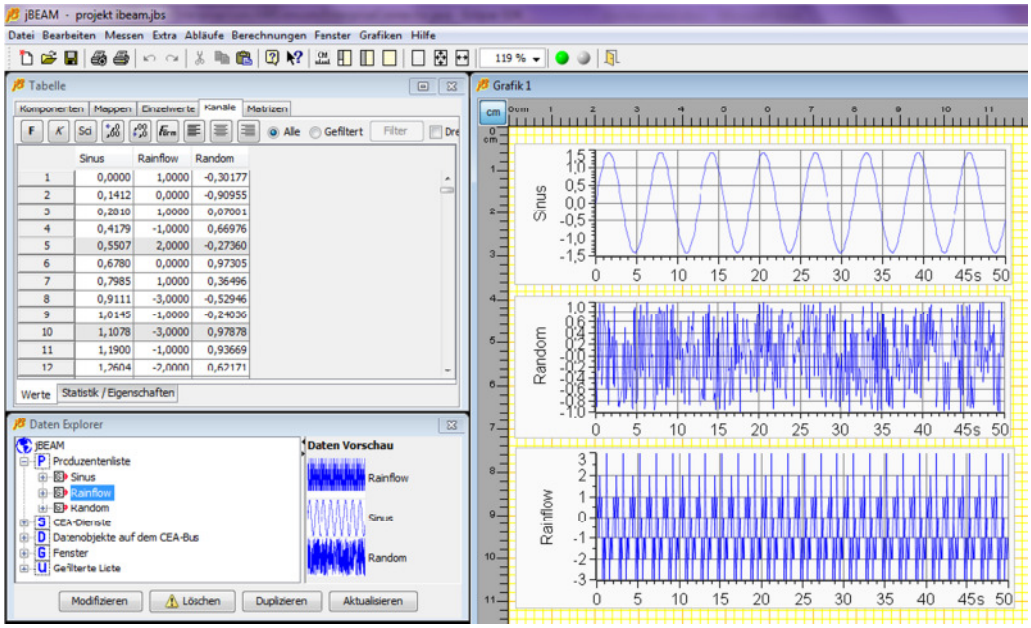


Abb. 2-2: Screenshot eines Beispielprojektes aus jBEAM

Der Startbildschirm von iBEAM bietet nun die Möglichkeit, sich mittels einer IP-Adresse und einem Port zu diesem jBEAM Server zu verbinden und sich die verfügbaren Kanäle anzeigen zu lassen.

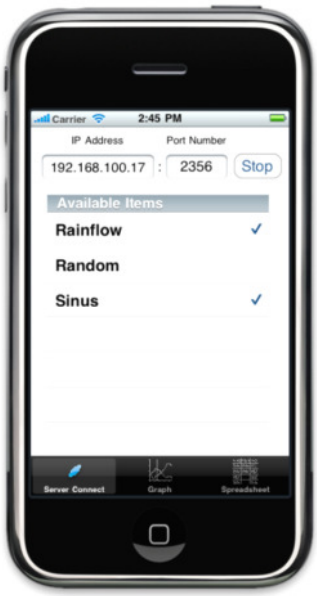


Abb. 2-3: Anmeldebildschirm von iBEAM mit abonnierten Kanälen (Quelle: Screenshot iPhone Simulator)

Ein Tippen auf den Kanal löst den Abonnier- oder Abmeldevorgang aus und ein abonniertes Objekt bekommt ein Häkchen angezeigt. Hierzu wird sich einer Tabelle bedient, welche auch mittels einer Wischgeste[2] durchscrollt werden kann.

Nur für abonnierte Objekte ist es möglich, sich den Graphen und die Werte anzeigen zu lassen. Dabei werden die Werte und Kanäle zwischen Server- und Client-Applikation synchron gehalten. Dies bedeutet, dass bei der Änderung von Werten oder auch Eigenschaften der Kanäle auf dem Server eine Event-Nachricht versendet wird, auf welche die Client-Applikation entsprechend reagiert und ihre Anzeigen aktualisiert.

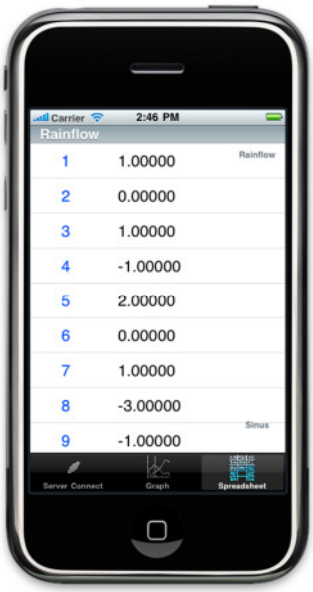


Abb. 2-4: Übersicht der Werte eines abonnierten Kanals in iBEAM (Quelle: Screenshot iPhone Simulator)

Wurden nun ein oder mehrere Kanäle abonniert, so ist es möglich sich deren Werte anzeigen zu lassen. Hierzu wechselt man mittels der *TabBar*[3] im unteren Bildschirmbereich auf den Tab „Spreadsheet“. Diese Ansicht listet die Werte eines Kanals in einer Liste auf, in welcher wieder mittels Wischgesten navigiert werden kann. Am rechten Bildschirmrand ist eine Liste mit den Namen aller abonnierten Kanäle zu sehen. Bei einem *Tip* auf einen dieser Namen wird zur Ansicht mit den Werten dieses Kanals gewechselt.

Der Tab „Graph“ führt zur graphischen Darstellung des Kanals. In dieser Darstellung kann mittels Touch-Gesten navigiert werden. So kann mittels der *Drag*-Geste der Graph verschoben werden und mittels der *Flick*-Geste, welches ein Stauchen oder Strecken mit

[2] Übersicht aller Gesten siehe Anhang 2
[3] Übersicht der Kontrollelemente siehe Anhang 6

zwei Fingern darstellt, kann man zoomen. In der rechten oberen Ecke befindet sich ein Button mit einem Pfeil, welcher dazu dient, den nächsten abonnierten Kanal anzuzeigen.

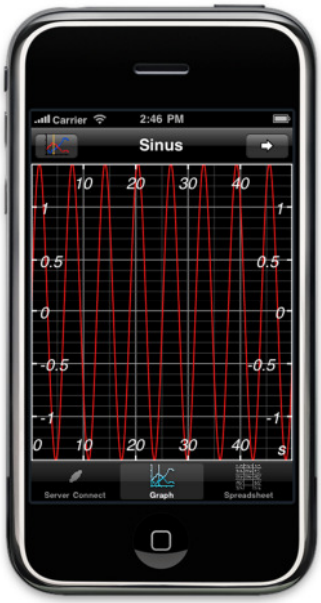


Abb. 2-5: Graph eines Kanals in iBEAM (Quelle: Screenshot iPhone Simulator)

Mittels des Tabs „Server Connect“ gelangt man nun wieder zum Startbildschirm, wo eine Übersicht aller verfügbaren Kanäle zu finden ist und die Möglichkeit besteht, diese zu abonnieren, sich von ihnen abzumelden oder auch die Verbindung zu beenden.

2.3 Funktionalität von iBEAM aus Programmiersicht

Dieses Unterkapitel soll einen groben Überblick über die programmiertechnische Struktur des Programms iBEAM geben. Implementierungsdetails, welche den Kern von iBEAM und somit das EnCom Protokoll betreffen, werden im nächsten Kapitel näher beleuchtet.

Die wichtigste Klasse für die Funktionalität von iBEAM ist der `DataItemManager`. Diese Klasse enthält eine Liste mit allen auf dem entfernten Server verfügbaren und von iBEAM unterstützten Items. Ein Item ist dabei eine Instanz einer Unterklasse von `AbstractDataObject`. Je nach Typ des Items kann es also ein Objekt der Klasse `DoubleChannel`, `DoubleValue`, `FloatChannel`, `FloatValue`, `IntegerChannel` oder `IntegerValue` sein. Demnach werden nur Werte oder eindimensionale Kanäle der Datentypen Float, Double und Integer von iBEAM unterstützt.

Der `DataItemManager` besitzt Methoden, um diese Items zu bearbeiten. Dazu gehört das Hinzufügen von Items oder das Entfernen. Außerdem können Änderungen an Items vorgenommen werden, wenn ein entsprechendes Events vom Server empfangen wird. Diese Änderungen werden außerdem an die graphischen Ansichten weitergeleitet, so dass diese sich aktualisieren können, um dem Anwender eine synchronisierte Sicht auf die Items zu gewähren.

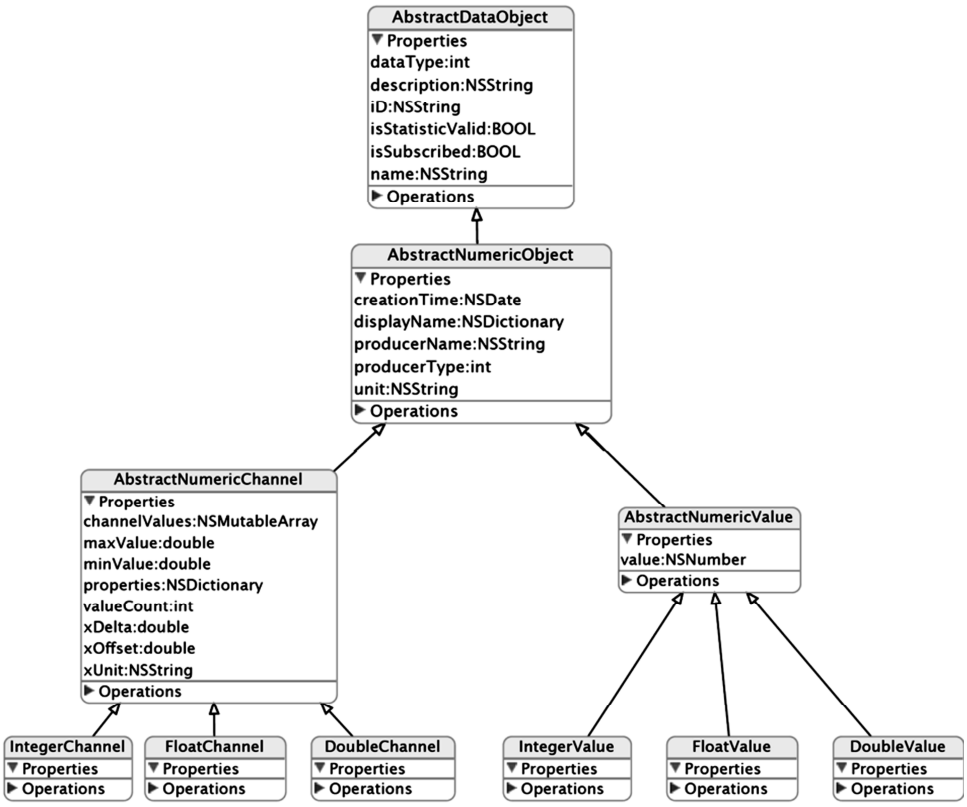


Abb. 2-6: Klassenhierarchie der Datenobjekte (Quelle: [Fritzler10] , S. 50)

3 Evaluierung des EnCom Protokolls

Das EnCom Protokoll wurde im Hause AMS entwickelt und stellt den Protokollablauf für das Produkt EnCom dar. EnCom als Produkt beinhaltet die Idee der Technologie, Implementierungen dieser Technologie und darauf aufsetzende Applikationen. Dabei steht EnCom für Enterprise Communication.

Das Protokoll dient hauptsächlich zur Netzwerkkommunikation von und mit den hauseigenen Produkten BEAM, jBEAM und iBEAM. Es legt den Satz und die Form der Nachrichten sowie die Vereinbarungen zur Kommunikation fest. Dabei wird auf eine einheitliche Implementierung und Nachrichtenstruktur gesetzt, welche in verschiedenen Programmiersprachen umgesetzt werden können. So sind zum Beispiel Implementierungen in Java und C# in der Protokollversion 3.0 vorhanden. Die jetzige Umsetzung in der Programmiersprache Objective-C auf Basis des Cocoa-Frameworks, wie sie im Produkt iBEAM zum Tragen kommt, weicht von der Struktur dieser anderen beiden Implementierungen ab und ist auch nur bis zur Protokollversion 2.3 kompatibel. Ein wichtiger Bestandteil dieser Arbeit wird sein, die Objective-C Implementierung an die Neuerungen der Protokollversion 3.0 anzupassen.

Im folgenden Kapitel wird zur Einstimmung auf die Entwicklung des Produktes und Protokolls eingegangen. Danach werden wichtige Prinzipien und Strukturen beleuchtet und auf die Vor- und Nachteile des Protokolls eingegangen, um die Bedeutung für diese Arbeit hervorzuheben. Die Informationen hierzu stammen aus der EnCom Präsentation [EnCom_Prä10] und den Dokumentationen zur Architektur [EnCom_Arch10] und zum Protokoll [EnCom_Prot10] der Firma AMS. Zum Abschluss des Kapitels soll die Objective-C Implementierung, welche von der Struktur her stark abweicht, zum Vergleich herangezogen werden.

3.1 Entwicklung des Produktes EnCom

EnCom wurde im Jahre 2007 entwickelt und war vorerst nur dafür ausgelegt, um mehrere Clients mit einem zentralen Server zu verbinden. Auf diesem Server wurden zum Beispiel Berechnungen ausgeführt. Die Resultate daraus konnten danach auf den Clients visualisiert werden. Diese Version lag nur in einer Java Implementierung vor und war fest in jBEAM integriert.

Im Jahre 2009 wurde mit iBEAM eine Applikation eingeführt, welche nur einen Teil des EnCom Protokolls unter der Programmiersprache Objective-C implementierte.

Bis zum Jahre 2010 wurden starke Strukturveränderungen und Erweiterungen an der in jBEAM integrierten Version vorgenommen und EnCom Version 2 entstand. Die wichtigste Erweiterung ist dabei die Möglichkeit der Lastverteilung auf mehrere Server. Auch wurde Wert darauf gelegt, das Protokoll Drittanbietern zugänglicher zu machen. Dazu wurde das

Protokoll von jBEAM losgelöst, so dass es auch eigenständig verwendet werden kann. Außerdem wurde die Nachrichtenstruktur verbessert und generischer gestaltet, um anpassbarer an individuelle Zwecke zu werden. Zu dieser Zeit wurde das Protokoll auch in die Sprache C# portiert, um mit dem Programm BEAM genutzt zu werden. Hierbei lag das Augenmerk darauf, die Klassenstruktur weitestgehend beizubehalten. Dank der ähnlichen Syntax zwischen Java und C# war es möglich, viel Quellcode zu portieren. Einige Java-Klassen wurden außerdem nachprogrammiert, um den Quellcode möglichst identisch zu halten.

3.2 Die Idee der EnCom-Technologie

Eine EnCom Applikation setzt sich aus drei Schichten zusammen. Dies ermöglicht eine höhere Flexibilität und Erweiterbarkeit, da diese Schichten einfach austauschbar sind. Diese drei Schichten sind die Netzwerkschicht, Nachrichtenschicht und die Logikschicht.

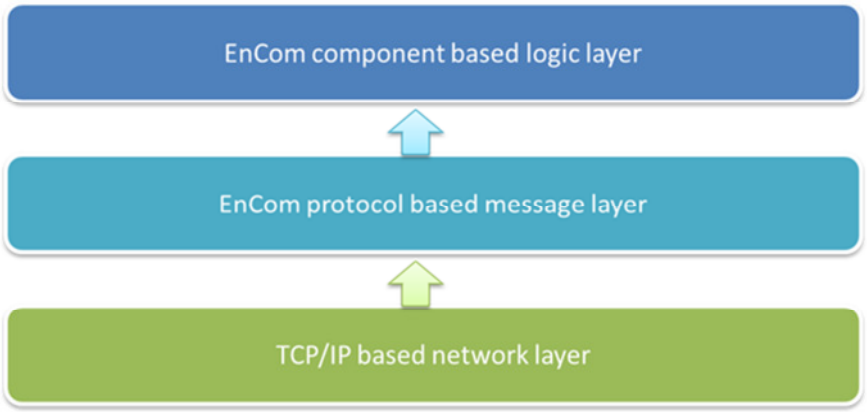


Abb. 3-1: EnCom Protokollstack (Quelle: [EnCom_Prä10])

Die Netzwerkschicht von EnCom setzt auf den TCP/IP Protokollstack auf und stellt Klassen zum Versenden und Empfangen von EnCom Nachrichten bereit. TCP ist ein verbindungsorientiertes Netzwerkprotokoll zur Übertragung von Daten. Dabei werden die Datenströme intern in Pakete zerlegt und verschickt. Es ist ein zuverlässiges Netzwerkprotokoll, da jedes Paket vom Empfänger bestätigt wird und verlorene Pakete noch einmal gesendet werden. Außerdem werden die Pakete beim Empfänger geordnet, so dass die vom Programm ausgelesenen Daten dem Datenstrom auf dem Absenderhost entsprechen.

Die Nachrichtenschicht definiert alle EnCom Nachrichten, welche zur Verfügung stehen. Dazu zählen Nachrichten zum Verbindungsaufbau, zur Datenabfrage oder zum Eventhandling. Diese Nachrichten sind dabei stark an das CEA Konzept angelehnt und eignen sich somit vor allem für messtechnische Zwecke. Jedoch besteht auch die Möglichkeit, mittels generischer Nachrichten Anfragen und Daten jeglicher Natur zu übertragen. Eine EnCom-Nachricht kann dabei als eine Folge von Werten verschiedener

Typen angesehen werden. Diese Typen und die Reihenfolge dieser sind für jede Nachricht fest definiert.

Die oberste Schicht ist die Logikschicht. Diese wird durch die jeweilige Applikation definiert und steuert die Abfolge und Auswertung der Nachrichten.

3.3 Einblick in die Implementierungen des EnCom Protokolls unter Java und C#

Die EnCom Protokoll- und Netzwerkschichten liegen bis jetzt in den beiden Programmiersprachen Java und C# vor. Dabei wurde darauf geachtet, dass eine weitestgehend identische Klassenstruktur eingehalten wurde, um Änderungen sehr schnell in die jeweils andere Sprache zu überführen. Der Grundgedanke ist somit, dass die gleiche Klasse auf den verschiedenen Systemen nach außen hin gleich reagiert.

Die aktuelle Protokollversion ist dabei die Version 3.0. Die wohl wichtigste Änderung zur Vorgängerversion 2.3 ist dabei die Übertragung der Nachrichten im Binärmodus und die optionale Komprimierung mittels ZLIB, welche die bisher vorhandenen Übertragungsarten Textmodus und GZIP-komprimiert ergänzt [EnCom_Arch10] . Dies brachte deutliche Verbesserungen der Performanz mit sich. Eine Übertragung der Nachrichten im Textmodus bedeutet, dass alle Werte in UTF8-String-Form in die Nachricht geschrieben werden. So wird zum Beispiel bei einer Gleitkommazahl jede einzelne Ziffer als UTF8-Zeichen in die Nachricht geschrieben und muss beim Empfänger als String ausgelesen werden und danach wieder zurück in eine Gleitkommazahl umgewandelt werden. Wird diese Gleitkommazahl im Binärmodus geschrieben, so wird ihre Bitdarstellung, so wie sie auch im Speicher vorhanden ist, geschrieben[4]. Dabei wird die *Endianness* beachtet und gegebenenfalls werden Bytes vertauscht. Alles in allem ist diese Methode jedoch deutlich schneller, da beim Empfänger nun die Gleitkommazahl binär ankommt und nicht noch einmal geparst werden muss.

3.3.1 Wichtige Klassen der Netzwerkschicht

3.3.1.1 DynamicDataBuffer

Der `DynamicDataBuffer` realisiert einen Serialisierungsmechanismus für Daten. Dabei stehen Methoden zur Verfügung, um alle primitiven Datentypen[5], eindimensionale Arrays, Datums- und Zeitangaben, Strings und Enums in ein sich dynamisch vergrößerndes

[4] Die Bitdarstellung folgt dem IEEE 754-1985 Standard [IEEE85]

[5] Wenn im Folgenden von primitiven Datentypen gesprochen wird, sind die primitiven Typen unter Java gemeint. Diese wären: boolean, byte, char, double, float, int, long und short. Eine Übersicht über diese ist im Anhang 1 zu finden.

Bytearray zu schreiben und daraus zu lesen. Da alle Objekte und deren Attribute auf primitiven Datentypen basieren und in diese zerlegt werden können, ist es möglich, jedes Objekt mittels dieser Methoden zu serialisieren. Die geschriebenen Daten können als Bytestrom ausgelesen und über das Netzwerk übertragen werden. Der `DynamicDataBuffer` ist also das Herzstück, um die Nachrichten in eine Bytefolge umzuwandeln, welche danach versendet werden kann.

Dabei ist es sehr wichtig, dass der `DynamicDataBuffer` auf jeder Plattform für gleiche geschriebene Werte, auch die gleiche Bytefolge erzeugt. So lautet eine Festlegung, dass die Bytefolge, welche erzeugt wird, immer in der *Big Endian Byte-Reihenfolge*[6] geschrieben wird, welche auch offiziell als *Netzwerkbyte-Reihenfolge* bezeichnet wird. Auch ist die Festlegung, dass die primitiven Datentypen auf allen Systemen mit der Größe geschrieben werden, welche sie auch in Java hätten. Dies bedeutet zum Beispiel, dass ein long-Wert immer mit acht Bytes geschrieben wird. Eine Übersicht der primitiven Datentypen unter Java ist im Anhang A zu finden.

3.3.1.2 EnterpriseConnector

Der `EnterpiseConnector` stellt die virtuelle Verbindung zu einem korrespondierenden `EnterpriseConnector` einer anderen Applikation her. Er besitzt Methoden, um sich zu einem Server zu verbinden oder sich selbst als Server zu starten.

Seine wichtigste Aufgabe ist jedoch das Senden und Empfangen von Nachrichten. Hierzu besitzt er Methoden um *Requests* und *Events* zu senden. Die Methode zum Senden eines *Requests* gibt auch gleich die vom entfernten Host kommende *Answer* zurück.

Er implementiert wiederum das Interface `EnterpriseConnectorIF`, welches all seine nach außen hin sichtbaren Methoden vorschreibt.

3.3.1.3 Interface EnterpriseMessageHandlerIF

Das Interface `EnterpriseMessageHandlerIF` stellt eine Schnittstelle für Klassen dar, welche der Ereignisbehandlung dienen. Es deklariert vier Methodenrümpfe:

- `handleMessage`
- `connectionClosed`
- `closeConnection` und
- `connectionEstablished`

[6] Weitere Informationen: [Intel10]

Das Interface bietet die Verbindung zur Applikation, welche auf das EnCom Protokoll aufsetzt. Je nach aufsetzender Applikation, können so Behandlungsroutinen für eingehende Nachrichten oder Verbindungsaufbau bzw. -abbau geschrieben werden.

Der `handleMessage`-Methode wird die zu behandelnde Nachricht mitgegeben. Diese kann ein *Request* oder ein *Event* sein. Antworten auf *Requests* werden nicht mit dieser Methode behandelt, da die Methoden zum Senden von *Requests* im `EnterpriseConnector` die *Answer* für diesen zurückliefern.

Die Methoden `connectionClosed` und `connectionEstablished` sollten jeweils Routinen zum Behandeln dieser Ereignisse beinhalten.

Für das Schließen der Verbindung soll die Methode `closeConnection` bereitstehen. Diese soll alle Routinen abhandeln, welche beim Schließen der Verbindung ausgeführt werden.

3.3.2 Klassen der Protokolls

Die Protokollschicht von EnCom enthält die Definitionen für die Nachrichten, welche übertragen werden können. Dabei werden die Möglichkeiten der Objektorientierung genutzt und die Nachrichten hierarchisch und über Interfaces realisiert.

Eine EnCom Nachricht, welche über das Netzwerk versendet wird, besteht im Groben aus zwei Teilen: einem *Header* und einem *Payload*. Der *Header* beinhaltet allgemeine Informationen über die Nachricht. Er umfasst eine feste Größe von 12 Bytes und beinhaltet folgende Informationen:

- Nachrichtentyp (Request, Answer oder Event)
- Nachrichtenform (Textnachricht oder binäre Nachricht)
- Eindeutige ID der Nachricht
- Komprimierung des *Payloads* (Unkomprimiert, GZIP-komprimiert oder ZLIB-komprimiert)
- Länge des *Payloads* in Bytes

Der *Payload* beinhaltet die eigentliche Nachricht in binärer Form, so wie sie über das Netzwerk versendet wird. Dieser ist so aufgebaut, dass die in der Nachricht definierten Parameter in einer festen Reihenfolge darin geschrieben werden. Dies muss so geschehen, damit die binäre Nachricht eindeutig auf den entfernten Host ausgelesen und interpretiert werden kann.

Es gibt ca. 30 Messages und mehrere Interfaces und abstrakte Klassen, welche in einer Vererbungshierarchie angeordnet sind.

So gibt es in der obersten Schicht dieser Hierarchie für jede Message-Art ein Interface, welches benötigte Methoden definiert. Somit werden einige grobe Message-Arten definiert,

wie zum Beispiel Answers-, Requests-, Event-, Status- oder Info-Messages. Diese sind untereinander wieder hierarchisch angeordnet, so dass z.B. alle Interfaces von dem Interface `IEnterpriseMessage` ableiten.

Danach folgt eine Schicht von abstrakten Klassen, welche diese Interfaces implementieren und allgemeine Methodenimplementierungen vorgeben, wie zum Beispiel Methoden zum Schreiben und Parsen des *Payloades*.

Die unterste Schicht dieser Hierarchie sind die konkreten Nachrichtenimplementierungen[7]. Diese enthalten eine statische, konstante *Map* mit Parameternamen und den dazugehörigen *Enterprise-Datentypen*[8]. Diese Mappe mit Parametern stellt die Informationen und Daten dar, die von der Nachricht übermittelt werden.

Abb. 3-2: Verbildlichung der Klassenhierarchie der EnCom-Nachrichten (Quelle: Eigene Darstellung)

Die Klasse `AbstractEnterpriseMessage` ist dabei eine abstrakte Klasse, welche die Oberklasse für alle abstrakten und konkreten Nachrichtenklassen[9] des EnCom-Protokolls darstellt.

[7] Übersicht der Nachrichten siehe Anhang 4
[8] Übersicht über Enterprise Datentypen siehe 0

Sie beinhaltet eine Mappe mit den Parameternamen und den dazugehörigen Parameterwerten einer Nachricht. Diese wird von den abgeleiteten, konkreten Nachrichten gefüllt, sobald diese instanziiert werden. Die konkreten Nachrichten besitzen wiederum eine statisch-konstante Mappe, welche den Parameternamen ihre *Enterprise-Datentypen* zuordnet.

So schreibt zum Beispiel eine Hello-Nachricht die Parameternamen (z.B. applicationName) und die dazugehörigen Werte (z.B. „iBEAM“) in diese Mappe. In der konkreten Hello-Nachrichtenimplementierung ist wiederum hinterlegt, dass zu dem Schlüssel „application_name“ der *Enterprise-Datentyp String* gehört, weil dieser Parameter immer ein String-Wert ist.

Soll diese Mappe nun in den *Payload* geschrieben werden, besitzt die Klasse `AbstraktEnterpriseMessage` Methoden, um Werte aller *Enterprise-Datentypen* in den *Payload* zu schreiben. Diese Methoden müssen so arbeiten, dass der geschriebene Wert unter Kenntnis des *Enterprise-Datentyps* eindeutig beim Empfänger ausgelesen werden kann. Die Methoden zum Auslesen und Parsen sind auch in der abstrakten Klasse implementiert. Aus dieser Mappe können nun wiederum die konkreten Message-Implementierungen die Werte für Parameter auslesen.

Somit übernimmt die Klasse `AbstraktEnterpriseMessage` alle Schreib-, Lese- und Parservorgänge.

Die folgende Abbildung stellt den Aufbau einer Hello-Nachricht dar.

Mappe in AbstractEnterpriseMessage		***statisch und konstante Mappe in konkreter Nachrichtenimplementierung***	
Parametername	**Parameterwert**	**Parametername**	**Parameterdatentyp**
protocol_version	„3.0.0“	*protocol_version*	String
application_name	„jBEAM“	*application_name*	String
application_version	„6.0.1.3“	*application_version*	String
application_user	„Administrator“	*application_user*	String
os_name	„Windows Vista“	*os_name*	String
os_version	„6.0“	*os_version*	String
architecture	„x86“	*architecture*	String
configuration	[["dataType", "STRING","Text"]]	*configuration*	Map

Form der Resultierenden Nachricht im Textübertragungsmodus:

```
Hello:protocol_version="3.0.0":application_name="jBEAM":
application_version="6.0.1.3":application_user="Administrator”:
os_name="Windows Vista":os_version="6.0":architecture="x86":
configuration=[["dataType","STRING","Text"]]
```

Abb. 3-3: Parameter am Beispiel einer Hello-Nachricht

3.4 Einblick in die EnCom Implementierung in iBEAM

Dieses Unterkapitel geht näher auf die Implementierung des EnCom Protokolls als Kern von iBEAM ein. Hierzu wird zuerst der Kommunikationsablauf untersucht, welcher zwischen iBEAM als EnCom Client und einem EnCom Server stattfindet. Danach werden weitere wichtige Merkmale beleuchtet und auch Unterschiede zu den Implementierungen in C# und Java aufgezeigt.

3.4.1 Der Kommunikationsablauf

Der Kommunikationsablauf von iBEAM mit der Serverapplikation entspricht dem normalen Kommunikationsablauf der meisten EnCom Anwendungen. Vorrausetzung ist, dass der EnCom Server gestartet wurde und bereit ist, neue Clients anzumelden. Die Initiative geht hierzu von der Clientapplikation aus, welche Informationen zur Anwendung und der unterstützten EnCom Protokollversion sendet und selbige Informationen für die Serverapplikation zurück bekommt. Die Konvention ist nun, dass die niedrigste Protokollversion der beiden Applikationen als Version für den folgenden Informationsaustausch herangezogen wird. Somit müssen die EnCom Implementierungen auch immer abwärtskompatibel zu älteren Protokollversionen sein.

Der Ablauf unterteilt sich dabei in verschiedene Phasen:

- Verbindungsphase: Die Client-Applikation verbindet sich zu einem Server. Es werden alle auf dem Server befindlichen Items abgerufen. Bei einer Protokollversion kleiner als 3.0, wurden dabei nur die IDs der Items übermittelt. Ab Version 3.0 werden auch Informationen, wie Name, Produzent und Datentyp übertragen.
- Informationen zu Items: Es werden die Informationen des Items, wie Produzent, Name und auch gegebenenfalls statistische Informationen übertragen.
- Arbeitsphase: In dieser Phase kann der Nutzer Items abonnieren. Dies bedeutet, dass der Client sofort über Änderungen der Daten des Items auf dem Server informiert wird.
- Trennung: Der Client trennt die Verbindung zum Server

Die folgende Abbildung liefert einen Überblick über den nachfolgenden Informationsaustausch:

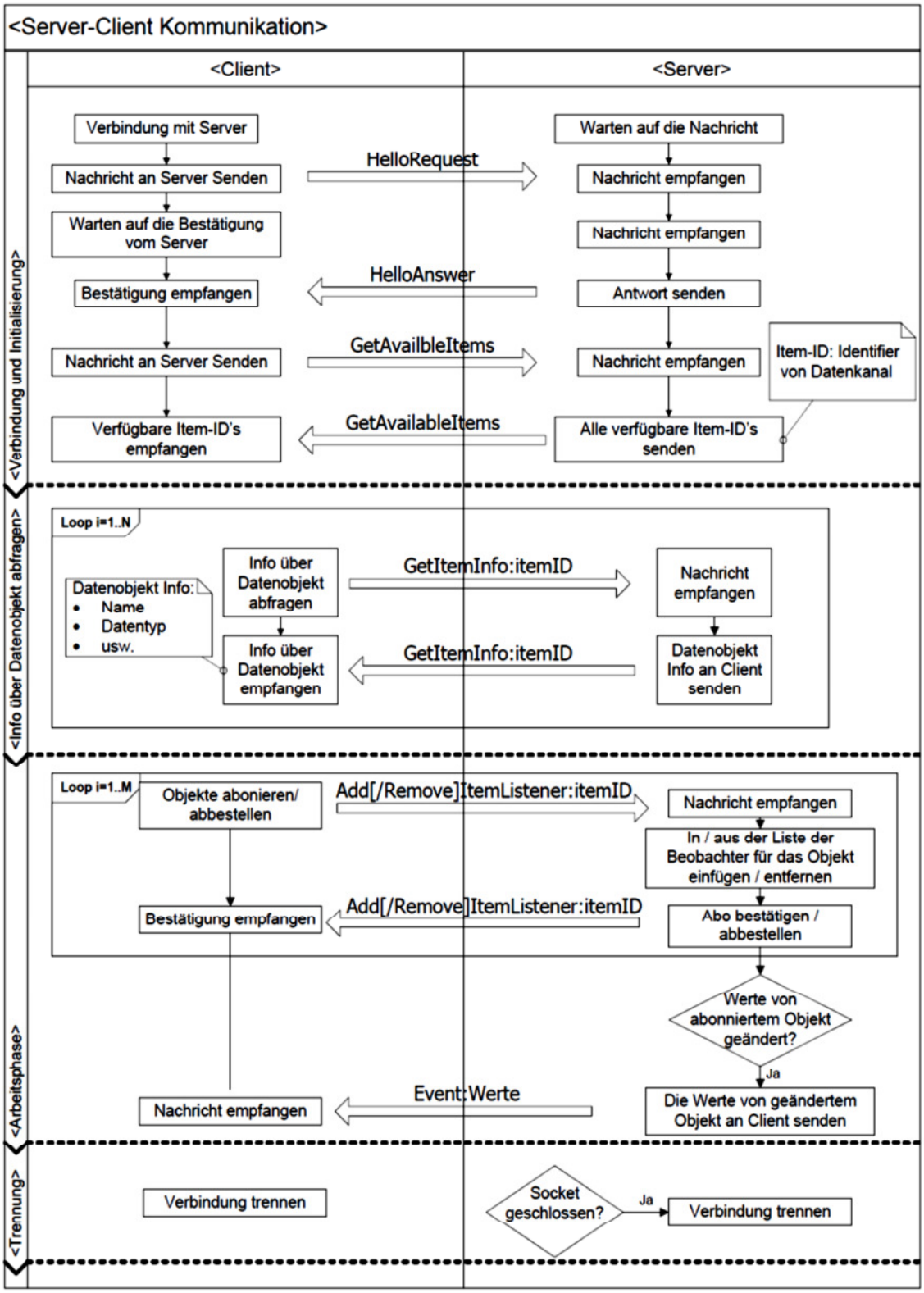


Abb. 3-4: Kommunikationsablauf eines EnCom Clients und Servers (Quelle: [Fritzler10] , S. 40)

3.4.2 Eigenschaften des iBEAM EnCom

iBEAM unterstützt zu Beginn der Arbeit die EnCom Protokollversion 2.3. Diese besitzt noch keine Datenübertragung im Binärmodus und ZLIB-Kompression, welche mittlerweile in der aktuellen Protokollversion 3.0 vorhanden sind. Diese sind auch nicht in der Implementation vorgesehen und wären schwer nachzurüsten. Es ist somit nur möglich, Daten in dem langsameren Textmodus und mit optionaler GZIP-Komprimierung zu übertragen. Der langsame Textmodus erleichtert dem Programmierer im Fehlerfall zwar den Debugging-Prozess, da die übertragenen Nachrichten für ihn „lesbar" sind, jedoch bringt es auch erhebliche Geschwindigkeitseinbußen mit sich, da so alle Werte aus dem String herausgelesen und in die entsprechenden Datentypen umgewandelt werden müssen. Da nummerische Werte der Hauptbestandteil der Übertragung zwischen dem EnCom Server und iBEAM sind, können diese schneller mit dem Binärmodus geschrieben, übertragen und ausgelesen werden.

EnCom unterstützt eine Funktion, welche bei der Übertragung von großen Kanälen mit mehr als 100.000 Werten zum Tragen kommt. Diese Funktion heißt *Channel Overview*. Dieser stellt eine Übersicht der Werte, welche in einem großen Kanal vorhanden sind, dar und kann zum Zeichnen verwendet werden. Der *Channel Overview* hingegen enthält 10.000 Maxima und Minima des Kanals. Zeichnet man nur diese, so unterscheidet sich die Graphik bei einem geringen Zoomfaktor nicht von der Graphik mit den originalen Werten. Erst bei einem relativ großen Zoomfaktor müssen die originalen Messdaten verwendet werden, damit die Graphik nicht verfälscht. Der *Channel Overview* kann jedoch nicht verwendet werden um die Werte eines Kanals in einer Tabelle anzuzeigen. Dafür gibt es die Nachricht `GetItemValues`, mit welcher die Werte in einem bestimmten Bereich angefordert werden können. Diese Nachricht wird von iBEAM jedoch nicht unterstützt und auch der *Channel Overview* wird in Nachrichten ignoriert.

Sieht man sich die Struktur von iBEAM in der Entwicklungsumgebung an, so fällt sofort auf, dass dabei viele Klassennamen des EnCom Protokolls auftauchen. Es fällt weiterhin auf, dass die Nachrichten zwar in einer ähnlichen, hierarchischen Struktur angeordnet sind, wie die Nachrichten der Java und C# Implementierungen, jedoch ist die Struktur innerhalb dieser Klassen unterschiedlich. So gibt es oft andere Methoden, welche auch anders arbeiten. Die Einarbeitung und das Nachvollziehen der Vorgänge fallen dabei schwer. Auch ist zu erkennen, dass sehr wenig Interfaces verwendet wurden. Das EnCom Protokoll setzt jedoch sehr auf Interfaces, um die Flexibilität und eine Austauschbarkeit von Klassen zu erhalten. So ist es zum Beispiel möglich, nicht den auf TCP/IP setzenden `EnterpriseConnector` zu verwenden. Stattdessen könnte selbst eine Klasse geschrieben werden, welche das Interface `EnterpriseConnectorIF` implementiert. Diese könnte dann eine eigene Technologie zur Kommunikation zu nutzen (z.B. UDP, SOAP, …).

Ein weiterer Aspekt der Objective-C EnCom Implementierung ist ihre starke Verbindung mit dem Programm iBEAM. Dies liegt dem Fakt zugrunde, dass sich kaum an die geschichtete Einteilung einer EnCom Applikation in Netzwerk-, Nachrichten- und Applikationsschicht gehalten wurde. Auch ist die iBEAM Implementierung bei weitem nicht so generisch gehalten, wie die anderen EnCom Implementierungen. So ist zum Beispiel der `EnterpriseConnector` im EnCom Protokoll dafür vorgesehen, Verbindung zum Server aufzubauen und Nachrichten zu übermitteln und zu empfangen. Hierbei sind die Methoden so generisch angelegt, dass die Parameter und Rückgabewerte Objekte sind, welche das oberste Interface `EnterpriseMessageIF` implementieren. Nimmt man nun die iBEAM Implementierung des `EnterpriseConnectors`, so ist zu sehen, dass dieser auch Methoden für den Verbindungsauf und -abbau bereitstellt, jedoch sind die Methoden zum Senden und Empfangen von Nachrichten nicht generisch. So gibt es für jeden Nachrichtentyp eine eigene Methode zum Senden und eine Empfangen dieser. Die Methoden zum Senden eines *Requests* geben auch nicht die Antwort als Rückgabewert zurück. Auch werden nicht alle Nachrichten betrachtet, welche das EnCom Protokoll bereitstellt. Das Nachrüsten dieser Nachrichten wäre somit aufwendig.

4 Strukturerneuerungen und Anforderungen an das neue Produkt

Aus den vorrangehenden Kapiteln ging hervor, dass iBEAM in seinem jetzigen Stand bei Weitem nicht den vollen Funktionsumfang des EnCom-Protokolls ausschöpft. Dieses Kapitel behandelt Konzepte und Argumente, um dies zu beheben und stellt einige Ideen zu Funktionserweiterungen von iBEAM dar. Danach werden einige Anforderungen an das Produkt iBEAM analysiert.

4.1 Argumente für eine Strukturerneuerung des iBEAM Kerns

Der Hauptgrund für die Strukturerneuerungen von iBEAM ist, dass zurzeit nur das EnCom Protokoll der Version 2.3 unterstützt wird und somit mit der aktuellen EnCom Protokollversion 3.0, welche bisher als Java- und C#-Implementierung vorliegt, nicht mehr stabil läuft. Bei kleinen Änderungen an den Java- und C#-Implementierungen muss immer darauf geachtet werden, dass die Kompatibilität zur iBEAM Implementierung gewahrt wird. Auch schöpft die Cocoa-Implementierung nicht den vollen Funktionsumfang des EnCom Protokolls aus. So werden Nachrichten nur im langsameren Textmodus und mit optionaler GZIP Komprimierung übertragen. Um eine performancetechnische Verbesserung zu erzielen, wäre die binäre Übertragung der Daten und die zusätzliche Komprimierung mittels ZLIB zu realisieren.

Des Weiteren sind in iBEAM nur wenige Nachrichten implementiert und es werden viele Nachrichten, die nicht unmittelbar für das Programm benötigt werden, ignoriert. Diese sind im Nachhinein schwieriger einzuführen, da nur eine sehr flache Nachrichten-Hierarchie vorhanden ist. Außerdem folgt die Struktur innerhalb der Klassen nicht denen des EnCom-Protokolls und somit ist eine Einarbeitung in dieser schwieriger möglich. Würde die Struktur des iBEAM Kerns an die hierarchische Struktur des EnCom Protokolls angepasst werden, könnten neue Nachrichten sehr einfach eingeführt werden.

Alles in allem ist die Erweiterung der Netzwerkkommunikation von iBEAM sehr aufwendig und eine Pflege des aktuellen Standes mühselig. Eine flexible und vollständige Implementation des EnCom-Protokolls ist somit wünschenswert und angestrebt.

Wäre nun das EnCom-Protokoll in seinem ganzen Funktionsumfang in Objective-C implementiert, würde dies völlig neue Möglichkeiten für das Produkt iBEAM eröffnen. Vor allem durch die Unterstützung weiterer Datentypen könnten völlig neue Einsatzspektren zum Tragen kommen, welche noch nicht absehbar sind. Das EnCom Protokoll könnte als eigenständige Bibliothek angesehen werden, welche auch in anderen Cocoa Anwendungen zum Einsatz kommen kann. Auch könnte es Kunden bereitgestellt werden, damit diese es für eigene Anwendungen nutzen könnten.

4.2 Konzepte der Strukturerneuerung des iBEAM Kerns

Für die Strukturerneuerung des Kerns von iBEAM gäbe es einerseits die Möglichkeit, die Regeln des EnCom Protokolls unter Objective-C mit eigenen Methoden und Konzepten nachzubilden, so wie es bisher in iBEAM geschah, oder die schon vorhandene Implementierung weiterzuführen.

Die Weiterführung der vorhandenen Implementierung wäre mit viel Einarbeitungszeit versehen und auch sehr schwierig und aufwendig zu vollziehen. Dies mit dem Resultat, dass die Implementierung sich immer noch deutlich von den Implementierungen in Java und C# unterscheiden würde und somit mit hoher Wahrscheinlichkeit weiterhin schwer zu warten bliebe. Damit ist der Aufwand einer völligen Neuimplementierung des EnCom Protokolls als Kern von iBEAM gerechtfertigt. Dies mit dem Hauptzweck eine solide, erweiterbare und leicht zu wartende Basis für die darauf aufsitzenden Programmfunktionalitäten zu gewährleisten. Die Portierung des EnCom Protokolls von Java auf C#, unter Einhaltung einer möglichst ähnlichen Klassenstruktur, hat sich auch in der Vergangenheit schon bewährt.

4.2.1 Ansätze für die Portierung des EnCom Protokolls auf Objective-C

Für eine Portierung des EnCom Protokolls auf Objective-C und das Cocoa Framework, unter Einhaltung einer möglichst ähnlichen Klassenstruktur, stehen mehrere Ansätze zur Verfügung.

Cross Compiler

Da die Implementierungen des EnCom Protokolls schon in den Programmiersprachen C# und Java vorhanden sind, gäbe es die Möglichkeit mittels Cross Compiler daraus unter Objective-C lauffähigen Code zu generieren.

Hierzu gibt es ein Projekt namens XMLVM [XMLVM10] , welches unter der GNU Public License entwickelt wird. Dies besitzt unter anderem die Möglichkeit aus Java Bytecode eine XML Datei eines bestimmten Formates zu erstellen. Diese XML Datei kann verwendet werden um daraus wiederum Objective-C Code zu erstellen. Auch ist es möglich eine lauffähige Anwendung zu erstellen.

Ein Nachteil dieser Methode ist die „Leserlichkeit“ des erzeugten Codes. Dies rührt daher, dass versucht wird die Vorgänge des Java-Bytecode nachzubilden. Ein Beispiel hierzu ist im Anhang 8 ein Beispiel für ein „Hello World“ zu finden.

Code Generation

Eine interessante Methode ist die Code Generation. Dies kann sich so vorgestellt werden, dass nach einer bestimmten Vorschrift, aus einem, vom Zielformat unabhängigen Format, Quellcode erzeugt wird. Dies geschieht mittels eines Code Generators.

Als ein Beispiel hierfür kann zum Beispiel die Umwandlung von Daten im XML-Format in ein HTML-Format mittels XSLT angesehen werden. Eine XSL-Datei stellt hierbei die Vorschrift (oder auch Template genannt) für die Erzeugung des Codes im HTML–Format dar. Der Code Generator ist dabei der XSLT-Prozessor.

Dieses Prinzip könnte auch für das EnCom-Protokoll genutzt werden. Somit könnten vor allem die konkreten Nachrichten im XML-Format definiert werden. Diese bestehen nur aus den Parametern, welche einem bestimmten Enterprise Datentyp angehören. Eine XML zur Definition einer Nachricht, müsste somit nur eine Einordnung der Nachricht in die Klassenhierarchie, die Parameter und deren Typen enthalten. Der Code Generator, welcher jedoch eigens entwickelt werden müsste, hätte nun die Aufgabe aus diesen XML Dateien der Nachrichten den Quellcode in den verschiedenen Programmiersprachen zu entwickeln. Der Vorteil dabei wäre, dass der Code Generator, bei entsprechender Programmierung auch Quellcode in verschiedenen Programmiersprachen erzeugen könnte, was wiederum auch einen erhöhten Aufwand bei der Programmierung des Code Generators bedeuten würde. Für die abstrakten Nachrichten oder andere Klassen des EnCom Protokolls wäre diese Methode jedoch ungeeignet, da diese Klassen oft umfangreiche Methoden besitzen, welche oftmals auch auf programmiersprachenspezifische Klassen zurückgreifen.

Somit ist diese Methode ungeeignet, da die ca. 30, sich selten ändernden Nachrichten des EnCom Protokolls den Aufwand für die Programmierung eines Code Generators nicht rechtfertigen. Außerdem ist der Aufwand für das Erstellen einer neuen Nachricht auch sehr gering, da die Hauptfunktionalität zum Parsen und Erstellen der Nachrichten in den abstrakten Oberklassen liegt.

Portierung von Hand

Die letzte Möglichkeit besteht in der Portierung von Hand. Dies bedeutet, dass jede Klasse von Hand portiert wird und an die syntaktischen Gegebenheiten unter Objective-C angepasst werden muss. Auch für die Klassen der Ausgangssprache müssten entsprechende Pendants des Cocoa-Frameworks gefunden werden. Diese Methode ist jedoch zu bevorzugen, da sie entscheidende Vorteile bietet:

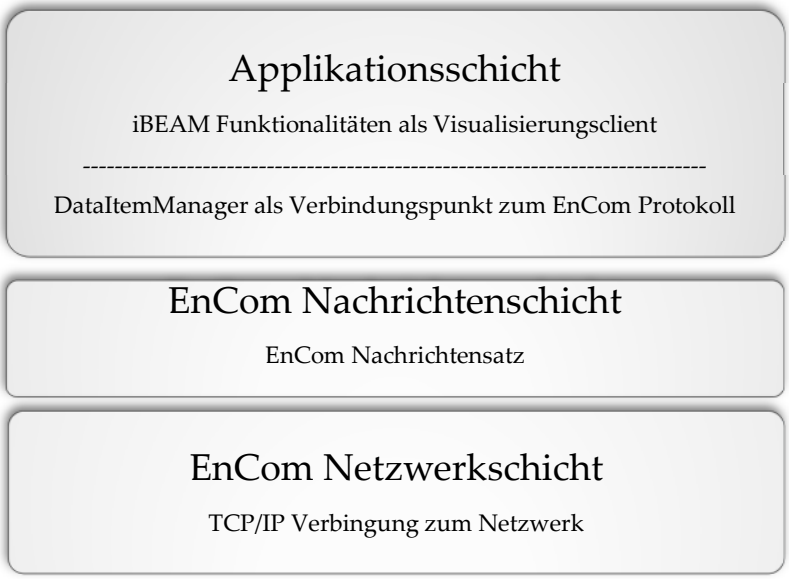
- Der erzeugte Code wird so weit wie möglich dem Java oder C# Code ähneln, was die Wartbarkeit des Produktes deutlich steigert. So könnten zum Beispiel Änderungen in der Java Implementierung sofort auf C# und Objective-C übertragen werden, da durch die gleiche Struktur die Stelle der Änderung leicht zu finden ist. Somit kann das Protokoll ohne viel Aufwand und Kenntnisse gepflegt werden.
- Der Code kann während des Programmierens auch mit Dokumentiert werden um so die Einarbeitungszeit für neue Mittarbeiter zu erleichtern.

- Das Protokoll bietet nach außen hin dieselben Schnittstellen, wie die C# und Java Implementierungen und erleichtert somit Kunden die Nutzung dieser Implementierungen, da sie nach außen hin gleich arbeiten.

Dies rechtfertigt die Portierung von Hand.

4.2.2 Aufbau des modernisierten iBEAM

Der Aufbau des modernisierten iBEAM soll nun vor allem so sein, dass das EnCom Protokoll unabhängig von der darüber liegenden Applikation ist. Hierzu soll die Hierarchie, welche auch in der EnCom Dokumentation [EnCom_Prä10] vorgeschlagen wird, eingehalten werden. Diese schlägt eine Dreiteilung vor. Hierzu gibt es eine Netzwerkschicht, welche für den Verbindungsaufbau und die Nachrichtenübertragung zuständig ist. Des Weiteren gibt es die Nachrichtenschicht, welche alle EnCom Nachrichten enthält. Die oberste Schicht soll die aufsitzende Applikation enthalten. Der `DataItemManager`, welcher auch schon in der alten iBEAM Implementierung vorhanden war, ist hierbei jedoch eine wichtige Klasse, welche die Verbindung zum EnCom Protokoll herstellt. Der `DataItemManager` besitzt eine Übersicht aller vorhandenen Items und verwaltet diese.



Der `DataItemManager` ist eine Klasse, welche Events und Anfragen der Applikation entgegen nimmt und passende Nachrichten hierzu versendet. Des Weiteren nimmt er aber auch EnCom Events entgegen und aktualisiert entsprechend seine Daten und leitet diese Events gegebenenfalls an die Applikation weiter. Der `DataItemManager` kann somit als Zwischenschicht zwischen Nachrichten- und Applikationsschicht angesehen werden.

4.3 Anforderungen an das Produkt iBEAM

iBEAM soll eine professionelle Anwendung darstellen, um zum Beispiel mittels Netzwerkkommunikation Messwerte von entfernten Prüfständen zu visualisieren oder Messungen zu steuern. Durch die vielfältigen Möglichkeiten des EnCom-Protokolls stehen diese auch dem darauf aufsetzenden Produkt iBEAM zur Verfügung. Das Ergebnis dieser Arbeit soll jedoch kein fertiges Endprodukt darstellen, sondern vielmehr ein Prototyp, welcher für Marketingzwecke an Kunden herangetragen werden kann um später mit Hilfe der Ideen und Ansprüche von Kundenseite erweitert zu werden.

4.3.1 Funktionale Anforderungen

Das Produkt iBEAM soll nicht an Funktionen einbüßen. Somit soll es weiterhin möglich sein, folgende Aktionen auszuführen:

- Verbinden zu einem entfernten EnCom-Server über TCP/IP
- Anzeigen verfügbarer, unterstützter Items auf dem Server
- Auf dem Server neu angelegte Items sollen angezeigt werden
- Abonnieren oder Abbestellen eines Items
- Abonnierte Items werden bei Änderungen auf dem Server auch geändert und somit synchron gehalten
- Anzeigen der Werte eines Items
- Visualisieren der Werte eines Items mittels eines Graphen

Zusätzlich sollen jedoch auch noch weitere Funktionen hinzukommen, welche noch nicht vorhanden sind:

- Es sollen nicht nur unterstütze Items angezeigt werden, sondern alle auf dem Server vorhandenen Items. Das Abonnieren darf jedoch nur für unterstütze Items möglich sein.
- Es soll eine Detailansicht zu jedem Item geben, in welcher Eigenschaften wie Informationen zum Produzenten oder statistische Daten zu sehen sind

4.3.2 Qualitätsanforerdungen

Im Folgenden werden die Qualitätsanforderungen an das Produkt iBEAM und die damit verbundene EnCom Implementierung gestellt. Diese Anforderungen werden mit den Gewichtungen „nicht relevant“, „normal“, „gut“ und „sehr gut“ bewertet.

Funktionalität

Die Funktionalität eines Softwareprodukts gibt Auskunft darüber, wie angemessen, sicher und richtig das Produkt arbeitet ist. Auch kann die Interoperabilität mit anderen Anwendungen dazu gezählt werden. Dieses Merkmal wird „gut“ eingestuft.

Zuverlässigkeit

Das Kriterium der Zuverlässigkeit beinhaltet die Fehlertoleranz, Reife und Wiederherstellbarkeit des Produktes. Da iBEAM nur als Visualisierungsclient eingesetzt wird und alle wichtigen Daten unverändert auf dem Server verbleiben und nicht geändert werden, kann dieses Kriterium mit „normal“ bewertet werden.

Benutzbarkeit

Die Benutzbarkeit eines Softwareproduktes gibt Aufschluss über den Grad der intuitiven, benutzerfreundlichen Bedienbarkeit und der Verständlichkeit des Programmes. Da sich das Produkt iBEAM vor allem durch seine intuitive Bedienung auszeichnen soll, wird dieses Merkmal mit „gut“ bewertet.

Effizienz

Das Merkmal Effizienz zeichnet sich durch das Zeit- und Verbrauchsverhalten aus. Dazu kann auch der Ressourcenverbrauch gerechnet werden. Dieses Merkmal kann mit einer „sehr guten“ Gewichtung eingeschätzt werden, da ein möglichst optimaler Ressourcenverbrauch auf den mobilen Touch-Geräten mit begrenzter Speichergröße und Prozessorgeschwindigkeit sehr wichtig ist. Weitere Informationen zu technischen Daten der Geräte im Anhang 7.

Änderbarkeit

Das Kriterium Änderbarkeit umfasst die Stabilität, Modifizierbarkeit und den Aufwand für die Ausführung von Verbesserungen. Dieses Kriterium sollte mit „sehr gut“ bewertet werden, da einerseits das EnCom Protokoll an sich schon dieses Kriterium fordert, da sich an diesem Protokoll Dinge ändern können, welche auch in iBEAM übernommen werden sollen, jedoch auch, da sehr häufig neue Versionen des *iOS* veröffentlicht werden, welche neue Features haben. Diese sollen leicht zu unterstützen sein.

Übertragbarkeit

Zu den Merkmalen der Übertragbarkeit zählen Kriterien, wie Anpassbarkeit, Installierbarkeit, Konformität und Austauschbarkeit. Dieses Kriterium ist schwer zu beurteilen, da das Produkt iBEAM nur für die Umgebung auf den iPhone, iPod Touch und iPad angedacht ist und nur auf diesen laufen kann. Auch spielt das Kriterium der Installierbarkeit keine Rolle, da in der Entwicklungsumgebung entwickelte Apps jeweils gleich leicht auf den Geräten zu installieren sind und man würde für diese Betrachtungswinkel ein „nicht relevant“ vergeben. Betrachtet man das Kriterium Austauschbarkeit jedoch im Zusammenhang mit dem iBEAM Kern, welcher das EnCom Protokoll darstellt, so würde dies mit „gut“ bis „sehr gut“ gewichtet werden, da es das Ziel ist, diesen Kern auch in anderen Applikationen einsetzen zu können.

5 Das Cocoa Framework und Objective-C

Die Programmiersprache Objective-C und das Cocoa Framework bieten die Grundlage, um für das iPhone programmieren zu können. Im folgenden Kapitel wird näher auf Besonderheiten dieser eingegangen um beim Leser die Grundlage für das Verständnis der folgenden Kapitel zu schaffen.

5.1 Das Cocoa Framework

Das Cocoa Framework (Weitere Informationen: [AppleRef10] , Cocoa Fundamentals Guide: What is Cocoa?) ist eine objektorientierte Programmierumgebung für Mac OS X und *iOS* und bildet die Grundlage zur Anwendungsentwicklung für diese Systeme. Es besteht aus objektorientierten Bibliotheken, einer Laufzeitumgebung und einer Entwicklungs-umgebung, namens Xcode. Es bietet die einzige Möglichkeit, für Apple Multi-Touch-Geräte Programme (auch Apps genannt) zu entwickeln.

Die Programmierumgebung Xcode

Xcode stellt eine in Mac OS integrierte Programmierumgebung dar. Damit ist es möglich, Projekte zu erstellen und zu verwalten. Der Editor bietet außerdem Syntaxhighlighting und die Möglichkeit, die darin geschriebenen Anwendungen zu testen und zu debuggen. Hierzu ist ein Simulator für das iPhone und iPad vorhanden.

Der Interface Builder

Der Interface Builder stellt einen eigenständigen Editor zum Erstellen und Bearbeiten graphischer Oberflächen für Mac OS und *iOS* Anwendungen dar. Hierbei stellt es nicht nur die Möglichkeit bereit, die graphischen Kontrollelemente auf die Oberfläche zu ziehen, sondern diese auch mit den Funktionalitäten aus dem sich in Xcode befindlichen Quellcode zu verbinden. Dies unterstützt auch das für Cocoa Oberflächenanwendungen vorherrschende Model-View-Controller Design Pattern[10]. Hierzu kann die graphische Oberfläche als „View“ im Interface Builder entworfen werden. Diese kann dann logisch mit den „Controllern“ verbunden werden. Diese werden aus der Entwicklungsumgebung Xcode genommen. Die „Model“-Klassen hingegen bleiben vor dem „View“ verborgen.

[10] Später in Kapitel 8.2.5

Performance Tools

Das Cocoa-Framework stellt außerdem noch einige Programme zur Verbesserung der Performance bereit. Das wichtigste darunter ist das Programm Instruments. Dies bietet verschiedenste Möglichkeiten, um während der Ausführung des Programms den Speicher, CPU-Auslastung oder Netzwerkzugriffe zu überwachen. So kann es vor allem sehr nützlich sein, um Speicherlöcher oder Performance-Engpässe aufzuspüren und zu beseitigen.

5.2 Einblick in die Programmiersprache Objective-C

Objective-C ist eine objektorientierte Programmiersprache, welche aus den Sprachen C und Smalltalk hervorgegangen ist. Die grundlegenden Eigenschaften unterteilen sich dabei einmal in die von der Sprache C geerbten Eigenschaften und die erweiterten Eigenschaften. Die folgenden Unterkapitel geben einen Einblick in die Syntax und gewisse Eigenheiten der Programmiersprache. Dabei wird sich zum Großteil auf die *iOS Reference Library* [AppleRef10] , *The Objective-C Programming Language* und das Buch *Programmieren fürs iPhone* [Stäuble09] bezogen.

5.2.1 Von C geerbte Eigenschaften

Objective-C umfasst eine Übermenge der Programmiersprache C. Dabei wird auf den offiziellen Standard ANSI-C gesetzt. Es können somit alle Funktionalitäten, welche die Sprache C bereitstellt, genutzt werden. Ergänzt werden diese durch objektorientierte Eigenschaften.

Header und Implementierungsdatein

So gibt es zweierlei Dateitypen. Einmal den die Header mit der Dateiendung „.h“, welche die öffentlichen Definitionen von Methoden (und in Objective-C der Klassen) beinhalten und die Implementierungsdateien mit der Endung „.m“ oder „.mm“[11], welche die Implementierung der Methoden und Klassen enthalten.

Datentypen

Es gibt 8 primitive Datentypen[12], welche aus der Sprache C übernommen wurden:

- `bool` für Wahrheitswerte
- `char` für Zeichen oder Ganzzahlen

[11] C Implementierungsdatein besitzen die Endung „.c“
[12] Übersicht im Anhang 1

- `short int`, `int`, `long int`, `long long int` für Ganzzahlen (vorzeichenbehaftet und vorzeichenlos)
- `double`, `float` für Gleitkommazahlen

Außerdem gibt es den „Nichtstyp“ `void`, welcher keine Werte speichert. Auch gibt es in Objective-C auch benutzerdefinierte Datentypen wie Strukturen, Unions und Enums. Diese bieten dem Nutzer die Möglichkeit, eigene Datentypen zu erstellen.

Enum

Ein Enum bietet somit die Möglichkeit, eine Liste von Aliasnamen für int-Werte zu erstellen. Diese können dann anstatt der int-Werte verwendet werden.

```
// Enum für Werktage
enum Werktage{
  Montag = 1,
  Dienstag,
  Mittwoch,
  Donnerstag
  Freitag
};
```

Listing 5-1: Beispiel für ein Enum mit Wochentagen

Strukturen

Strukturen stellen zusammengehörige Werte dar, welche zu einem Typ zusammengefasst werden. Sie können als Klassen ohne Methoden angesehen werden.

```
struct datum{
  short tag;
  char *monat;
  short jahr;
};

// Beispiel um den Tag zu setzen
struct datum d;
d.tag = 12;
```

Listing 5-2: Beispiel einer Struktur für ein Datum

Unions

Unions dienen zur Speicherung mehrerer Werte unterschiedlicher Datentypen im selben Speicherbereich. Somit ist eine unterschiedliche „Sicht“ auf einen Speicherbereich möglich. Dabei wird für die Union so viel Speicher reserviert, wie der größte Datentyp benötigt.

```
union converter{
  int int_;
```

```
    char bytes[4];
};

// int-Wert in Union
union converter con;
con.int_ = 6;

// Zugriff auf Bytes
con.bytes;
```

Listing 5-3: Beispiel einer Union zur Ermittlung der Byterepräsentation eines int-Wertes

Zeiger

Die wohl auffälligste Eigenschaft ist die Verwendung von Zeigern. Diese können als einen Zeiger oder Adresse zu einem Bereich im Speicher angesehen werden. Ein Zeiger auf einen Werte eines Datentyps wird folgendermaßen gekennzeichnet:

```
Datentyp * // (z.B. int * kennzeichnet einen Zeiger auf einen
// int-Wert)
```

Es gibt verschiedene Verwendungsmöglichkeiten für diesen Operator. Einmal kann er dazu verwendet werden, um bei Funktionsaufrufen nur einen Zeiger auf das Element zu übergeben. Vor allem bei großen Datenstrukturen bringt dies eine Verbesserung der Performance mit sich. Da nicht die ganze Struktur auf den Aufrufstack gelegt wird, sondern nur ein Zeiger auf das Element. Eine weitere Verwendung findet bei dynamisch allokierten Arrays Anwendung. Die Größe von dynamisch allokierten Arrays steht erst zur Laufzeit fest und sie werden mit der Funktion `malloc` oder `calloc` erzeugt. Diesen Funktionen wird die Größe des zu allokierenden Speicherbereiches mitgeteilt und sie liefern daraufhin einen Zeiger auf den Beginn dieses Speicherbereiches zurück. Dieser allokierte Speicherbereich muss, wenn er nicht weiter benötigt wird, mit der `free` Funktion freigegeben werden.

5.2.2 Erweiterte Eigenschaften

Die von C geerbten Eigenschaften erweitert Objective-C noch durch objektorientierte Eigenschaften. Somit unterstützt Objective-C objektorientierte Ansätze wie Polymorphismus, Methodenüberschreibung und dynamisches Binden.

Der Datentyp id

Der Datentyp id kann als generischer Datentyp angesehen werden, welcher Objekte jeden Typs beinhalten kann.

Definition der Klassen

Die Definition des Klasseninterfaces findet in der Headerdatei statt. Dort werden nach außen hin sichtbare Methoden und alle Instanzvariablen deklariert. Private Methoden müssen auf einem anderen Weg deklariert werden.

Eine das Interface einer Klasse wird in Objective-C wie folgt im Header definiert:

```
@interface Klassenname : Oberklasse <Interfacees>{
  // Deklaration der Instanzvariablen
  int a;
  }
  // Methodendeklatation
  - (void) doSomething;
@end
```

Listing 5-4: Definition einer Klasse im Header

Die Implementierung der Klasse erfolgt in der Implementierungsdatei. Dort werden die Funktionalitäten für die Methoden definiert.

Die Implementierung der Klasse sieht so aus:

```
#import "header.h"

// Kategorie der Klasse, welche zur Deklaration privater
// Methoden genutzt wird
@interface Klassenname (private)
  -(void) privateMethod;
@end

@implementation Klassenname : Oberklasse
  // Methodendeklaration
  - (void) doSomething{
      // ...
  }

  -(void) privateMethod{
      // ...
  }
@end
```

Listing 5-5: Implementierung einer Klasse

Dabei ist zu sehen, dass im oberen Teil des Ausschnitts noch einmal eine Definition des Klassen Interfaces stattfindet. Dies geschieht um somit private und nach außen hin nicht sichtbare Methoden zu definieren. Hierzu bedient man sich einer Spracheigenschaft, welche sich *Kategorie* nennt. Eine *Kategorie* kann verwendet werden um einer Klasse

Funktionalitäten hinzuzufügen. In diesem Beispiel wird sie jedoch zur Deklaration privater Klassen verwendet.

Protokolle

Protokolle können mit den Java-Interfaces verglichen werden. Diese dienen dazu um eine Gruppe von Methoden zu deklarieren, welche von der Unterklasse implementiert werden müssen. Hierbei ist zu beachten, dass ein Verstoß dagegen nur eine Compilerwarnung hervorruft und keinen direkten Fehler. Protokolle werden ähnlich wie Klassen definiert, nur dass hierbei das Schlüsselwort nicht `@interface`, sondern `@protocol` lautet.

Deklaration und Aufruf von Methoden

Methoden werden einmal im Header deklariert und in der Implementationsdatei definiert. Die Deklaration folgt dabei folgendem Muster:

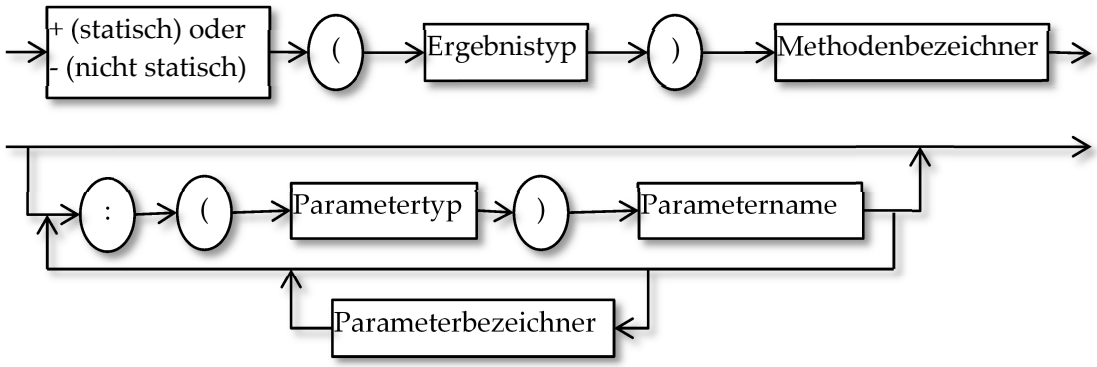


Abb. 5-1: Syntax der Methodendeklaration in Objective-C (Quelle: Eigene Darstellung)

Der Aufruf der Methoden wird im Zusammenhang mit Objective-C oft als „senden einer Nachricht“ bezeichnet. Das senden einer Nachricht bedeutet, dass das Objekt zum Ausführen einer Methode aufgefordert wird. Die zu sendende Nachricht wird dabei mit eckigen Klammern umschlossen. Sie und folgt der Reihenfolge von Empfänger, Nachricht (Methodenselektor) und den Parametern der Nachricht, wenn es welche geben sollte.

Hieraus ergeben sich folgende Beispiele:

```
// statische Methode zum Addieren zweier Ganzzahlen
+ (int) addiereA: (int) a mitB: (int) b;
// Methodenname: addiereA:mitB
// Aufruf:
[Klasse addiereA: 1 mitB: 2];

// statische Methode zum Addieren zweier Ganzzahlen ohne
// Parameterbezeichner
+ (int) addiere: (int) a : (int) b;
// Methodenname: addiere::
```

```
// Aufruf:
[Klasse addiere: 1 : 2];

// Ein Objekt aus einer Collection zurückgeben lassen
- (id) getObjectForKey: (int) key;
// Methodenname: getObjectForKey:
// Aufruf:
[object getObjectForKey: 3];

// Methode ohne Rückgabewert und Parameter
// Methodenname: updateAnsicht
- (void) updateAnsicht;
//Aufruf:
[object updateAnsicht];
```

Listing 5-6: Beispiel für Methodendeklaration und -aufruf unter Objective-C

Das Überladen der Methoden ist in der Form nicht möglich, da es nicht erlaubt ist in einer Klasse nicht mehrere Methoden mit gleichem Namen, aber unterschiedlichen Parametertypen existieren dürfen. Der Name einer Methode besteht aus dem Methodenbezeichner und den (optionalen) Parameterbezeichnern durch „:“ getrennt.

5.3 Speicherverwaltung unter Cocoa

Die folgenden Unterkapitel berufen sich auf den Apple Memory Management Guide [Ref10] und Kapitel 24 – Memory Management des Buches *Learn Objective-C for Java Developers* [Bucanek09] .

5.3.1 Was ist Memory Management?

Speicher ist eine Ressource, welche nicht unbegrenzt zur Verfügung steht und muss somit entsprechend verwaltet werden. Dabei ist es wichtig, Speicher, welcher nicht mehr benötigt wird, wieder frei zu geben und so wieder verfügbar zu machen, damit er von anderen Instanzen genutzt werden kann. Die Methoden dafür nennt man Memory Management.

5.3.2 Arten des Memory Management

Für Mac OS X Anwendungen gibt es die Möglichkeit des automatischen Memory Managements mittels Garbage Collection. Für das *iOS* gibt es diese Möglichkeit nicht und der Programmierer muss sich selbst um die Verwaltung seines Speichers kümmern.

Garbage Collection

Bei Garbage Collection handelt es sich um Mechanismen, welche sich eigenständig um die Freigabe des Speichers kümmern. Vom Nutzer angelegte Objekte werden selbstständig aus dem Speicher entfernt, wenn sie nicht mehr benötigt werden. Moderne Garbage Collection

Mechanismen beruhen auf der Erstellung eines Erreichbarkeitsgraphen für alle lebenden Objekte. Alle Objekte, die darin nicht enthalten sind, werden bereinigt.

Speicherallokation und -freigabe

In der Programmiersprache C wird das Prinzip der Speicherallokation und –freigabe verfolgt. Dabei werden wird Speicher mittels der Funktionen `malloc`, `calloc` und `ralloc` allokiert und mit der Funktion `free` freigegeben. Dies muss von Nutzer selbst getan werden, wenn ein allokierter Bereich nicht länger benötigt wird. Ansonsten kann es passieren, dass der Verweis auf diesen Bereich verloren geht wird und es nicht mehr möglich ist, diesen freizugeben oder ihn zu nutzen. Dies wird Memory Leak oder Speicherleck genannt.

Diese Methode birgt einige Gefahren. Einerseits kann es zu den schon angesprochenen Speicherlöchern kommen. Es kann jedoch auch geschehen, dass mehrere Parteien einen Zeiger auf einen gemeinsam genutzten Speicherbereich besitzen. Wird dieser Bereich von einer Partei freigegeben, ist dieser somit auch für die andere Partei verloren. Greift die Partei auf diesen freigegebenen Speicherbereich zu, kann dies zu fatalen Fehlern führen.

5.3.3 Referenzzählen unter Objective-C

Das Objective-C Memory Management setzt auf die Funktionen des Allokieren und Freigeben des Speichers unter C auf. Auch hier ist es so, dass allokierter Speicher freigegeben werden muss um, Speicherlecks zu vermeiden. Der Fortschritt besteht jedoch darin, dass die Methode des Referenzzählens angewendet wird. Jeder Besitzer eines Objektes zählt den Referenzzähler dieses Objektes bei der Übernahme des Besitzes hoch. Benötigt ein Besitzer das Objekt nicht mehr, gibt er es frei und der Referenzzähler des Objektes wird dekrementiert. Somit stellt der Referenzzähler immer die Anzahl der Besitzer eines Objektes dar. Erreicht der Referenzzähler Null, wird der Speicherbereich des Objektes sofort freigegeben. Dies geschieht durch den internen Aufruf der `dealloc`-Methode des Objektes. Diese kann überschrieben werden und sollte alle Instanzvariablen freigeben und Netzwerk-Verbindungen schließen.

5.3.3.1 Beeinflussung des Referenzzählers

Der Referenzzähler eines Objektes wird beim Allokieren auf eins gesetzt und zeigt somit an, dass es einen Besitzer gibt. Dies ist derjenige, der das Erstellen des Objektes beauftragt hat. Auch kann der Referenzzähler direkt mit dem Senden einer `retain`-Nachricht hochgezählt werden. Dies kann zum Beispiel der Fall sein, wenn ein Objekt von mehreren Instanzen genutzt wird.

Beim Aufruf der `release`-Methode wird der Referenzzähler dekrementiert und sofort die `dealloc`-Methode aufgerufen, wenn dieser Null ist. Die `dealloc`-Methode trägt dazu bei, dass das Objekt aufgeräumt und dessen Speicher freigegeben wird.

Des Weiteren gibt es noch die Möglichkeit, Objekte zu einem Autorelease Pool hinzuzufügen. Dies kann man sich als eine Sammlung von Verweisen auf Objekte vorstellen. Ein Objekt wird zu einem Autorelease Pool hinzugefügt, wenn es die `autorelease`-Message erhält. Wird der Autorelease Pool geleert, wird an alle enthaltenen Objekte die `release`-Message gesendet. Wird dadurch der Referenzzähler eines Objektes auf Null gesetzt, wird es aufgeräumt. Meist ist nicht genau zu sagen, wann ein Autorelease Pool geleert wird. So ist es sinnvoller, direkt eine `release`-Message zu senden, wenn man sicher ist, dass das Objekt nicht mehr benötigt wird. Somit kann der Speicher direkt freigegeben werden, was auf speicherarmen Systemen, wie dem iPhone, von Vorteil ist.

Jede Applikation sollte einen Autorelease Pool besitzen. Bei Benutzung des Apple Application Kit zur Erstellung graphischer Oberflächen, wird am Anfang jeder Ereignis-Abarbeitung, wie zum Beispiel nach einem Touch-Event auf einen Button, ein Autorelease Pool angelegt und am Ende dieser geleert. Somit braucht sich der Nutzer in den meisten Fällen nicht eigenhändig um die Erstellung kümmern. Es ist jedoch auch in einigen Fällen sinnvoll oder nötig, im Verlauf der Programmabarbeitung Autorelease Pools anzulegen. Diese wären:

- *Bei der Abarbeitung eines Threads:* Jeder Thread benötigt einen eigenen Autorelease Pool, denn Objekte, welche während der Abarbeitung des Threads „autoreleased“ werden, kommen in den threadeigenen Pool. Ist kein Autorelease Pool vorhanden, gibt es keine Möglichkeit mehr, diese Objekte aus dem Speicher zu bereinigen.
- *Bei Schleifen und Programmteilen, welche viele temporäre Objekte anlegen:* Werden viele temporäre, autoreleased Objekte erzeugt, können diese viel Speicherplatz belegen und würden erst in unbestimmter Zukunft aufgeräumt werden, wenn der nächste Autorelease Pool geleert wird. Die Lösung dafür ist, vor der Erzeugung der Objekte einen eigenen Pool anzulegen und diesen nach der Abarbeitung zu leeren.

5.3.3.2 Regeln der Speicherverwaltung unter Objective-C

Die oberste Regel bei der Speicherverwaltung unter Objective-C lautet:

Nur an Objekte, welche man besitzt, darf eine release- oder autorelease-Nachricht gesendet werden.

Hierzu ist zu klären, was den Besitz eines Objektes definiert. Es gibt zwei Möglichkeiten ein Objekt zu besitzen:

- Man legt ein Objekt mittels einer Methode, welche mit `alloc` oder `new` beginnt oder welche `copy` enthält
- Man sendet eine `retain`-Message an ein Objekt

Um den Besitz eines Objektes zu beenden, sendet man eine `release`- oder `autorelease`-Nachricht an ein Objekt. Dabei ist der einzige Unterschied, dass `autorelease` eine

release-Nachricht in der Zukunft bedeutet, wenn der Autoreleasepool geleert wird. Aus diesen Regeln ergeben sich einige zu beachtende Programmiermuster.

Methoden, welche mit alloc oder new beginnen oder ein copy enthalten, geben immer Objekte zurück, welche einen Referenzzähler von eins besitzen. Dies bedeutet, dass der Aufrufer der Methode der neue Besitzer des zurückgegebenen Objektes ist und somit auch verantwortlich dafür, seinen Besitz zu beenden, wenn er das Objekt nicht weiter benötigt.

Methoden, welche nicht mit alloc oder new beginnen und kein copy enthalten, müssen Objekte mit einem ausgewogenen Referenzzähler zurückgeben. Dies bedeutet, dass der Zähler theoretisch Null sein müsste. Da diese sofort aus dem Speicher bereinigt würden, bevor sie zurückgegeben werden könnten, wird an diese Rückgabeobjekte eine autorelease-Nachricht gesendet. Dies bewirkt, dass die Objekte erst später bereinigt werden. Und zwar, wenn der aktuelle Autoreleasepool geleert wird.

```
- (id)someObject{
   id object = [[[NSObject alloc] init] autorelease];
   …
   return object; // autoreleased object
}
```

Listing 5-7: Methode, welche ein Objekt mit ausgeglichenem Referenzzähler zurückgibt. (Quelle: [Bucanek09] , S. 442, Listing 24-9)

Wenn so ein ausgewogenes Objekt vom Aufrufer der Methode, auf längere Zeit genutzt werden soll und der Aufrufer neuer Besitzer des Objektes werden mag, muss an dieses Objekt eine retain-Nachricht gesendet werden.

Auch bei Methoden, welche einer Instanzvariablen einen Methodenparameter zuweisen, muss beachtet werden, dass somit diesem Parameter ein neuer Besitzer hinzugefügt werden muss. Dies wird mit dem Senden der retain-Nachricht getan.

```
- (void)setOne:(id)object
{
    [one autorelease];
   one = [object retain];
}
```

Listing 5-8: Methode, welche einer Instanzvariable einen Methodenparameter zuweist

5.3.4 Zusammenfassung

Die Verwaltung des Speichers ist auch in Zeiten, wo es immer mehr davon gibt, wichtig. Sie kann einerseits von Mechanismen wie dem Garbage Collection übernommen werden oder aber der Nutzer muss selbst die Verantwortung dafür übernehmen. Dabei ist zu beachten, dass Garbage Collection auch wieder Zeit benötigt um ausgeführt zu werden. Die performantere Lösung ist also, wenn der Nutzer sich selbst um die Verwaltung des

Speichers kümmert. Dies kann jedoch auch häufiger zu Fehlern führen und darum ist die Einhaltung einiger wichtiger Grundregeln Voraussetzung, um die optimale Nutzung dieser Ressource zu gewährleisten.

6 Implementierungsdetails für das Portieren des EnCom Protokolls

Dieses Kapitel beschäftigt sich mit ausgewählten Problemen und Entscheidungen, welche bei der Portierung des EnCom Protokolls von Java und C# auf Objective-C und Cocoa auftraten.

6.1 Wrapperklassen für primitive Datentypen und Arrays dieser

In Objective-C gibt es 8 primitive Datentypen: `bool`, `char`, `int`, `short`, `long`, `long long`, `double`, `float`. Die ausschließliche Benutzung dieser für die Speicherung von Zahlen und dynamischen Arrays in Form von Zeigern auf das erste Element birgt einige Nachteile und Risiken. In den folgenden Unterkapiteln wird auf die Problematik von primitiven Datentypen und dynamischen Arrays eingegangen und eine Lösung in Form von Wrapperklassen[13] aufgezeigt, die durch einige Quelltextauszüge unterlegt werden soll.

6.1.1 NSNumber als Hüllenklasse des Cocoa-Frameworks

Primitive Datentypen bringen einige Probleme mit sich. So können diese zum Beispiel nicht in Kollektionen, wie Listen oder Sets gespeichert werden, da diese nur Container für Objekte darstellen. Dafür bietet das Cocoa Framework eine Lösungen in Form der Klasse `NSNumber` an. Diese Klasse kann als Container für die primitiven Typen dienen. So wird eine Instanz dieser Klasse, welche einen float-Wert speichert, wie folgt angelegt und ausgelesen:

```
NSNumber *wrappedFloat = [NSNumber numberWithFloat: 2.3f];
float primitiveFloat = [wrappedFloat floatValue];
```

Listing 6-1: Kapselung eines float-Wertes in einem NSNumber-Objekt

Objekte der Klasse `NSNumber` können nun in Kollektionen gespeichert werden. Um herauszufinden, welcher primitive Typ ursprünglich in einem `NSNumber`-Objekt gespeichert wurde, gibt es eine Methode namens `objCType`, welche einen String in Form eines Charakter-Arrays[14] zurückgibt. Dieser String entspricht dem Wert der Compilerdirektive `@encoding()` des Datentyps. Diese Strings können verglichen werden.

```
NSNumber *wrappedNumber = [NSNumber numberWithFloat: 2.3f];

char *typeString = [wrappedNumber objCType];
```

[13] Auch Hüllklassen genannt
[14] Sogenannte C-Strings

```
if(strcmp(typeString, @encoding(int)) == 0)
     int primitiveInt = [wrappedNumber intValue];
else if(strcmp(typeString, @encoding(float)) == 0)
     float primitiveFloat = [wrappedNumber floatValue];
```

Listing 6-2: Bestimmung des Datentyps, welcher in einem NSNumber-Objekt gekapselt ist

Wie am obigen Code-Beispiel zu erkennen ist, ist es recht umständlich, den primitiven Datentyp, welcher hinter dem `NSNumber`-Objekt steht, herauszufinden.

6.1.2 Arrays von primitiven Datentypen

Es gibt mehrere Möglichkeiten, Arrays von primitiven Datentypen in Objective-C zu gestalten. Einmal gibt es Methoden, welche von der Sprache C übernommen wurden. Dies sind statische und dynamische Arrays (auch Felder genannt).

Die Feldgrenzen von statischen Arrays müssen schon zur Übersetzungszeit festliegen. Dies bedeutet, dass schon der Programmierer die genaue Größe des Arrays wissen muss. Im Gegensatz dazu wird die Größe von dynamischen Arrays zur Laufzeit bestimmt. Diese Arrays werden mit der *malloc()* Funktion angelegt. Dieser wird die Größe des Speicherblocks, welcher für das Array reserviert werden soll, mitgegeben. Zurückgegeben wird ein Zeiger auf den Beginn des Bereiches und somit das erste Element. Dynamische Arrays werden auf dem Heap des Speichers abgelegt und müssen im Gegensatz zu den auf dem Stack gelagerten statischen Arrays wieder freigegeben werden.

Für die meisten Arrays gilt, dass die eigentliche Größe zur Kompilierzeit noch nicht bekannt ist und somit dynamische Arrays verwendet werden. Diese haben im Gegenzug zu statischen Arrays einen großen Nachteil. Ihre Größe ist nicht mit der *sizeof()*-Funktion zu bestimmen. Die Größe eines dynamischen Arrays muss also immer in einer separaten Variablen mitgereicht werden. Auch ist es so, dass diese dynamischen Arrays nicht vom Objective-C Speichermanagement in Form von Referenzzählung profitieren können. Nehme man einmal an, ein Array von int-Werten würde von verschiedenen Instanzen des Programmes genutzt werden. Benötigt eine Instanz dieses Array nicht mehr, wäre sie dazu verpflichtet, den Speicherbereich des Arrays freizugeben, indem sie die *free()*-Funktion aufruft. Dies würde dazu führen, dass der Bereich bereinigt würde und auch für die anderen Instanzen, die dieses Array nutzen, verloren wäre. Andererseits sollte der Speicherbereich wirklich freigegeben werden, wenn niemand mehr dieses Array nutzt. Da es aber keinen Referenzzähler für diese primitiven Arrays gibt, welcher die Anzahl der aktuellen Besitzer angibt, kann dieser Zeitpunkt nur sehr kompliziert bestimmt werden.

Das Cocoa-Framework bietet somit eine eigene Datenstruktur zur Repräsentation eines Arrays an. Die Klassen *NSArray* und *NSMutableArray*. Diese bieten die Möglichkeit, Objekte in einer arrayähnlichen Datenstruktur zu Speichern. Der Unterschied der beiden Klassen besteht darin, dass Objekte der *NSArray*-Klasse nach der Initialisierung nicht mehr

verändert werden können. So ist es nicht mehr möglich, Objekte hinzuzufügen oder zu entfernen. Dies ist hingegen bei der Klasse *NSMutableArray* möglich. Auch sind Arrays dieser Klasse veränderlich in ihrer Länge. Diese beiden Klassen stellen somit Datenstrukturen für eine Sammlung von Objekte dar, auf die mit Hilfe eines Index, zugegriffen werden kann. Dabei können Objekte verschiedenster Typen in ein Array dieser Art gespeichert werden. Dies wird mittels des generischen Datentyps *id* realisiert und jedes Objekt, welches zurückgegeben wird, muss in den entsprechenden Typ gecastet werden. Um nun wieder primitive Datentypen in ein Array dieser Art zu speichern, muss man sich wieder der Klasse *NSNumber* bedienen und auch wieder deren Methoden zur Bestimmung des eigentlichen Datentyps verwenden, was wiederum aufwendig ist.

6.1.3 Argumente für die Erstellung von eigenen Wrapperklassen für primitive Datentypen

Wie in den vorhergehenden Unterkapiteln beschrieben, bietet das Cocoa-Framework Möglichkeiten zur Umhüllung primitiver Datentypen mit Objekten, welche auch für die Speicherung in Kollektionen genutzt werden können. Jedoch ist der Weg zur Bestimmung des Datentyps, welcher dahinter steht, relativ aufwendig[15]. Abhilfe hierzu sollen eigene Wrapperklassen bieten. Dazu wird zu jedem primitiven Datentyp eine eigene Klasse benötigt, die eine Instanzvariable dieses Typs besitzt.

Diese werden objektorientiert hierarchisch angeordnet und haben die gemeinsame, abstrakte Oberklasse *AbstractDataType*, welche die abstrakte Methode zur Erstellung eines Strings besitzt.

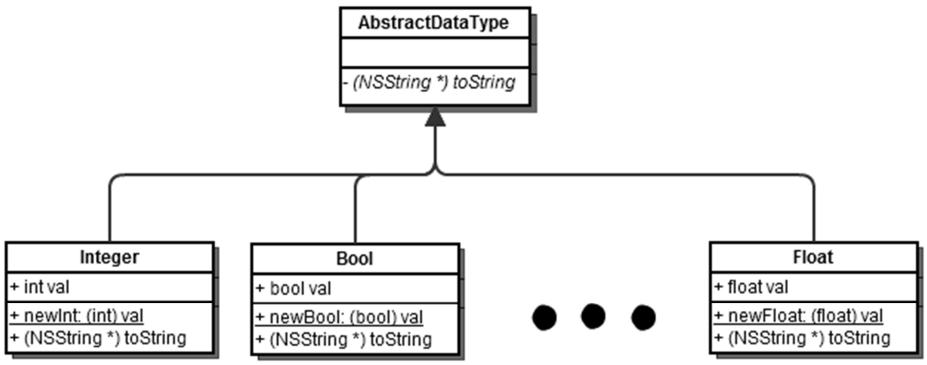


Abb. 6-1: Veranschaulichung der Wrapperklassen für primitive Datentypen

[15] Siehe Listing 6-2

6.1.4 Argumente für die Erstellung eigener Wrapperklassen für Arrays von primitiven Datentypen

Da nun schon Hüllklassen für die primitiven Datentypen existieren, wäre nun auch die Speicherung dieser in den bereitgestellten Klassen *NSArray* bzw. *NSMutableArray* einfacher. Es muss nicht mehr abgefragt werden, welchen Datentyp das zurückgegebene *NSNumber*-Objekt denn nun in Wirklichkeit beherbergt. Dazu können die eigens erstellten Wrapperklassen verwendet werden. Jedoch wurde sich nichtsdestotrotz für eigene Implementierungen von Arrayklassen für die primitiven Typen entschieden. Dies hat mehrere Gründe.

Hierzu sei zu erwähnen, dass in den EnCom-Nachrichten Implementierungen sehr oft auf Arrays primitiver Datentypen zurückgegriffen wird. Am häufigsten finden hierzu ein-, zwei- und dreidimensionale Arrays Verwendung. Diese Arrays haben eine feste Länge und beinhalten nur Werte ein und desselben Typs. Sie werden auch sehr oft als Rückgabewerte oder Parameter für Methoden genutzt. Die Nutzung der, nach dem primitiven Typen benannten, Wrapperklassen ist somit auch für den Programmierer günstiger, da er gleich am Typ des Rückgabewertes oder Parameters ablesen kann, von welchem Typ die Werte in dem Array-Objekt sind. Auch wird gewährleistet, dass wirklich nur Werte desselben Typs in das Array gespeichert werden, da Instanzen der Klasse *NSArray* grundsätzlich Objekte jeden Typs beherbergen.

Ein weiterer, wichtiger Grund ist, dass die vom Cocoa-Framework bereitgestellten Klassen ein anderes Verhalten im Vergleich zu normalen Arrays aufweisen. Die Klasse *NSArray* stellt somit eine Datenstruktur bereit, auf deren beinhaltende Objekte später nur noch zugegriffen werden kann, welche jedoch nicht verändert werden dürfen. Auch die Klasse NSMutableArray repräsentiert ein Array, welches keine Begrenzung hat. So können immer weitere Objekte hinzugefügt werden und der Programmierer muss sich selbst darum kümmern, dass die Arraygrenzen nicht überschritten werden. Beim Entfernen eines Objektes ist es so, dass die Indices der nachfolgenden Objekte dekrementiert werden. Somit entspricht das Verhalten eines NSMutableArrays eher dem einer Liste.

6.1.5 Implementierung der Wrapperklassen

Die Wrapperklassen für die Primitiven Datentypen werden sehr einfach aufgebaut. So gibt es für die primitiven Datentypen und eindimensionalen Arrays jeweils eine abstrakte Oberklasse, welche Attribute, die in allen Unterklassen vorhanden sind und Methoden, welche sie implementieren müssen, vorgibt. So gibt es die Klasse `AbstractDataType`. Diese gibt als einziges die Methode `toString` vor. Diese kann später gut benötigt werden um die gekapselten Werte in String Form darzustellen. Der gekapselte Wert wird jeweils als Instanzvariable in den konkreten Klassen definiert. Außerdem beinhaltet jede Klasse Methoden um den Wert abzufragen und einen Konstruktor.

Die Klasse `AbstractDataType1D` stellt die Oberklasse für die Arrayklassen dar. Diese enthält eine Instanzvariable für die Arraylänge. Das Array in sich wird jedoch in den konkreten Klassen gekapselt. Dies geschieht mittels eines C-Arrays. Dies stellt einen allokierten Bereich in der Größe des Arrays dar und einen Zeiger auf das erste Element dieses Arrays. Der Speicherbereich wird im Konstruktor allokiert. Wird dem Konstruktor ein Array mit Werten übergeben, so werden diese in das interne Array kopiert. Dies ist sicherer, weil so der Fall ausgeschlossen werden kann, dass das übergebene Array an anderer Stelle freigegeben wird, während es noch in der Klasse genutzt wird. Hierzu sei noch einmal zu erwähnen, dass für C-Arrays nicht die Memorymenagementregeln gelten und diese keinen retain-Zähler besitzen. Die konkreten Klassen besitzen nun außerdem noch Methoden um auf die einzelnen Elemente des Arrays zuzugreifen.

6.2 Der DynamicDataBuffer

Der `DynamicDataBuffer` stellt das Herzstück für die Serialisierung der zu übertragenden Daten dar. Hierzu bietet er Methoden für das Schreiben und Lesen von primitiven Datentypen, eindimensionalen Array, Strings, Datums- und Zeitangaben und Enums und erzeugt daraus ein Byte-Array, welches über das Netzwerk übertragen werden kann. Am anderen Ende kann dieses Byte-Array wiederum in einen `DynamicDataBuffer` geschrieben werden. Danach können daraus dieselben Werte, welche auf der sendenden Applikation geschrieben wurden, gelesen werden. Es ist somit von äußerster Wichtigkeit, dass die Daten so geschrieben werden, dass sie eindeutig wieder herausgelesen werden.

6.2.1 Die Java-Implementierung

Die Java-Implementierung des `DynamicDataBuffers` verwaltet intern eine Instanz der Klasse `ByteBuffer`. Der `ByteBuffer` kann als ein Bytearray angesehen werden und stellt diverse Methoden bereit, um in diesen zu schreiben und daraus zu lesen. Ein wichtiges Merkmal ist jedoch, dass dieser `ByteBuffer` eine feste Größe hat und diese nicht überschritten werden darf. Der `DynamicDataBuffer` hingegen soll sich dynamisch erweitern, wenn über seine derzeitige Größe hinaus geschrieben werden soll. Dazu ersetzt er den internen `ByteBuffer` durch einen Größeren und überträgt die Daten des alten Buffers in den Neuen.

Die Klasse `ByteBuffer` besitzt grundsätzlich die Möglichkeit die Daten in der Big oder Little Endian *Byte-Reihenfolge* zu schreiben. Das EnCom Protokoll legt jedoch fest, dass die Daten in den `DynamicDataBuffer` und somit auch den internen `ByteBuffer` in der Big Endian *Byte-Reihenfolge*, da dies die *Byte-Reihenfolge* ist, welche von der *Java Virtual Machiene* genutzt wird und auch den Standard für die Übertragung über das Netzwerk darstellt. ([Intel10] , S. 7) Dies muss somit auch in den Implementierungen in anderen Programmiersprachen der Fall sein.

6.2.2 Implementierung unter Objective-C

Für die Implementierung des `DynamicDataBuffers` unter Objective-C wurde sich dafür entschieden, die Struktur von Java weitestgehend zu übernehmen. Dazu zählt also die Implementierung der Klassen `ByteBuffer` und `DynamicDataBuffer`.

Der `ByteBuffer` besitzt intern ein Bytearray mit einer bei der Initialisierung festgelegten Größe (Kapazität genannt). Des Weiteren werden Zeiger in Form von `int`-Werten, welche Indices in dem Bytearray darstellen, auf wichtige Stellen in diesem Array benötigt. Diese wären aktuelle Position, das Limit und ein Marker.

Die aktuelle Position zeigt an, an welcher Stelle als nächstes geschrieben oder gelesen werden soll. Sie wird nach dem Lese- bzw. Schreibvorgang um die Anzahl der geschriebenen oder gelesenen Bytes inkrementiert. Dies bedeutet, dass der Buffer immer von vorn nach hinten durchgearbeitet wird.

Das Limit des `ByteBuffers` stellt die Position dar, bis zu welcher geschrieben oder gelesen werden kann. Dies muss nicht zwingend der Kapazität entsprechen. Das limit darf von außen gesetzt werden.

Der Marker dient dazu, eine bestimmte Stelle im Buffer zu markieren. Dies geschieht mit der `mark`-Methode und wird zum Beispiel bei der `reset`-Methode benötigt. Dabei wird der Zeiger der aktuellen Position zurückgesetzt auf die markierte Position.

Des Weiteren werden noch einige weitere Methoden benötigt, welche diese Zeiger beeinflussen. So gibt es eine `flip`-Methode. Diese wird zum Beispiel aufgerufen, wenn vollkommen in den Buffer geschrieben wurde und er nun bereit ist, um daraus zu lesen. Dabei wird das Limit auf die aktuelle Position gesetzt, da nicht darüber hinaus gelesen werden darf. Die aktuelle Position wird danach auf den Anfang des Buffers zurückgesetzt, um ihn auf das Lesen vorzubereiten.

Nun gibt es außerdem `put`- und `get`-Methoden für die primitiven Datentypen und Bytearrays. Diese bewirken das Schreiben oder Lesen eines Wertes dieses Typs und das inkrementieren des Positionszeigers[16].

Hierbei ist jedoch die *Byte-Reihenfolge* zu beachten. Die in den `ByteBuffer` geschriebenen Bytes müssen im Big Endian Format gespeichert werden. Auf dem iPhone werden Daten jedoch im *Little Endian* Format im Speicher abgelegt. Dies bedeutet, dass das Byte mit dem niederwertigen Bits (in binärer Schreibweise also die Bits, welche am weitesten rechts

[16] Bytegröße der Datentypen ist im Anhang 1 zu finden.

stehen) zuerst in den Speicher gelegt werden. *Big Endian* handhabt dies genau andersherum.

Nehme man so zum Beispiel eine 32 Bit Integerzahl mit der binären Repräsentation:

01010101 00000000 11110000 11111111

Diese würde nun wie folgt im Speicher abgelegt:

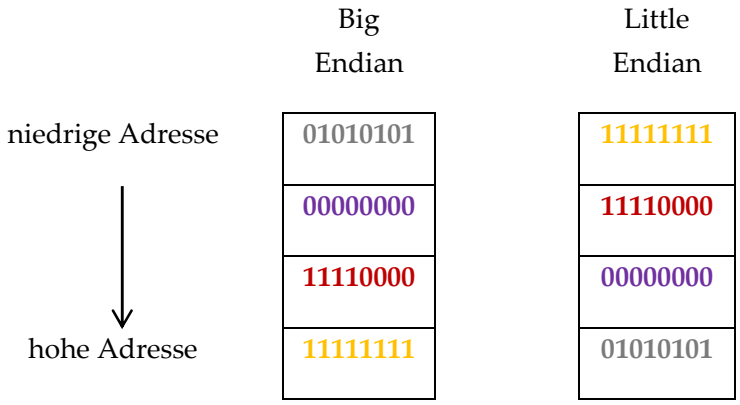


Abb. 6-2: Big und Little Endian Speicheransicht (Quelle: Eigene Darstellung)

Dies bedeutet, dass die Bytes getauscht werden müssen, bevor sie in den ByteBuffer geschrieben werden. Hierzu gibt es nun mehrere Möglichkeiten.

Es gibt die c-Funktionen `ntohl`, `ntohs`, `htonl` und `htons`. Diese bieten eine Konvertierung von Netzwerkreihenfolge zu Hostreihenfolge (net to host) und umgekehrt (host to net) für netlong- und short-Werte (32 und 16 bit). Netzreihenfolge bedeutet dabei *Big Endian* und die Hostreihenfolge hängt vom jeweiligen System ab. Hierzu ist jedoch zu sagen, dass mit dieser Methode nur 16- und 32-bit Zahlen umgewandelt werden können.

Eine weitere Möglichkeit besteht in der Nutzung von Unions. Dies waren Strukturen, die die Sprache C bereitstellt, um einen Container bereitzustellen, in denen sich Variablen unterschiedlicher Datentypen einen Speicherbereich teilen. Dies ist zur Konvertierung geeignet. Hierzu wird eine Union folgender Struktur angelegt:

```
typedef swapunion{
  double double_;
  float float_;
  int int_;
  short short_;
  long long long_;
  char bytes[8];
} swapunion;
```

Listing 6-3: Union zum Umwandeln von Datentypen zwischen Big und Little Endian

Nun kann man wie folgt auf die Bytes eines double-Wertes zugreifen:

```
// Beiden Unions zum Drehen der Bytes
swapunion beforeSwap;
swapunion afterSwap;

// Double Wert (Rechner ist Little Endian)
double littleEndianDouble = 2.4;
// Wert wird in die Union eingetragen
swap.double_ = d;
// Variable, die Anzahl der Bytes eines Doublewerts speichert
int doubleSize = sizeof(double);

// Drehen der Bytes
for(int i = 0; i < doubleSize; i++)
  afterSwap.bytes[doubleSize - i] = beforeSwap.bytes[i];

// Räpresentaion des Doublewertes auf einem Big Endian System
double bigEndianDouble = afterSwap.double_;
```

Listing 6-4: Beispiel zum Umwandeln eines Doublewertes von Little in Big Endian

6.2.3 Test

Dieses Kapitel zeigt exemplarisch einen Test auf, um sicherzustellen, dass der `DynamicDataBuffer` unter Java und Objective-C gleich arbeitet. Der zugehörige Quelltest ist in Null zu finden.

Gegenstand des Testes ist die Übertragung eines Float-Arrays von einer Testapplikation unter Objective-C (Client) zu einer Testapplikation unter Java (Server). Ein Float-Array bietet sich hierzu an, da so einerseits die Schreibroutinen für Arrays geprüft werden und andererseits das Schreiben von Gleitkommazahlen. Arrays werden so geschrieben, dass zuerst die Länge des Arrays geschrieben wird und danach folgen die einzelnen Werte. Der Datentyp Float wurde gewählt, da die Byterepräsentation von Float-Werten komplexer ist als die Darstellung von Ganzzahlwerten.

Der Ablauf sieht dabei so aus, dass sich die Client Applikation zu der schon laufenden Serverapplikation verbindet und ein Array mit Floatwerten sendet. Dazu wird das Floatarray in den `DynamicDataBuffer` geschrieben und die daraus resultierenden Bytes versendet. Diese werden auf dem Server ausgelesen, mittels eines `DynamicDatabuffers` wieder in ein Floatarray umgewandelt und auf der Konsole ausgegeben. Daraufhin wird dieses Array wieder in den Buffer geschrieben und die resultierende Bytefolge zurück an den Client gesandt, welcher diese wiederum ausliest und das Array auf der Konsole ausgibt. Anhand des Ausgangsarrays und der beiden Konsolenausgaben ist zu erkennen, dass der `DynamicDataBuffer` für ein Floatarray auf den Java und Objective-C Systemen gleich reagiert.

Dieser Test wurde für alle primitiven Datentypen, deren Arrays und Strings wiederholt und wurde erfolgreich bestanden. Bei späteren Tests mit der Übermittlung von EnCom-Nachrichten wurde auch noch einmal bestätigt, dass der unter Objective-C programmierte `DynamicDataBuffer` eine konforme *Byte-Reihenfolge* erzeugt.

7 Performance-Testszenarien für die neue EnCom Implementierung

Dieses Kapitel dokumentiert einige Testszenarien, welche die Performance der neuen EnCom Implementierung testen sollen.

7.1 Vorraussetzungen

Die folgenden Tests wurden von dem iPhone Simulator der Entwicklungsumgebung Xcode heraus ausgeführt. Hierzu wurde im iPhone Simulator eine Testapplikation laufen gelassen, welche die EnCom Implementierung unter Objective-C nutzt um Nachrichten an einen EnCom Server auf einem entfernten Rechner zu schicken.

Als EnCom Server wird hierfür jBEAM Version 6.3.0.2 genutzt.

7.2 Pingtest

Dieser Test testet den Durchsatz der neuen EnCom Implementierung. Hierzu werden 10.000 Ping-Nachrichten übertragen und die Zeit dazu gemessen. iBEAM sendet dabei jeweils immer einen `PingRequest` und wartet, so wie es im EnCom Protokoll vorgesehen ist, die Antwort ab, um danach den nächsten *Requst* zu senden.

Hierbei werden die komprimierten Übertragungsarten außer Acht gelassen, da erst Nachrichten mit einem *Payload* größer als 10 kB werden komprimiert. Die Ping Nachricht erreicht diese Größe nicht.

Versuchsanordnung:

Verwendete EnCom Nachricht: Ping

Anzahl der Ping-Nachrichten: 10.000

Netzwerkgeschwindigkeit: 100 Mbit/s

Versuchskonfiguration	Durchlauf 1	Durchlauf 2	Durchlauf 3	Mittelwert
Text, unkomprimiert	11,87	11,56	11,51	11,65
Binär, unkomprimiert	11,85	11,10	11,11	11,35

Abb. 7-1: Resultate des Ping-Tests

Auswertung:

Anhand der Messungen ist zu erkennen, dass kein gravierender Unterschied zwischen der Übertragung der Ping-Nachricht im Binär- oder Textmodus. Dies rührt daher, dass nur der *Payload* in Text oder Binärform geschrieben wird. Für die Header spielt dies keine Rolle. Der *Payload* der Ping Nachricht enthält nun hingegen nur den String „Ping“ und somit ist

es unerheblich, ob die Nachricht im Binär- oder im Textmodus geschrieben wird, da in jedem Fall nur dieser String geschrieben und ausgelesen wird.

7.3 Übertragung eines Kanals mit vielen Daten

Dieser Test untersucht das Verhalten beim Übertragen einer großen Nachricht mit Double-Werten. Hierbei werden einerseits die Übertragungsmodi binär und Text untersucht, aber auch die Komprimierung der Nachrichten.

Versuchsanordnung:

Für diesen Versuch werden von iBEAM aus mehrmals hintereinander die Werte eines Double-Kanals mit 10.000 Werten abgerufen. Hierzu wird die GetItemValues-Nachricht genutzt, welcher die Anzahl der Werte und die Blocknummer mitgegeben werden kann. Block Null bedeutet hierbei, dass der Block bei dem Wert mit dem Index Null beginnt.

Übertragener Kanal: Double Kanal mit 10.000 Werten

Verwendete EnCom Nachricht: GetItemValues, mit Block Nr. 0 und 10.000 Werten

Der Versuch wurde zweimal durchgeführt, wobei der erste Versuch über eine ungedrosselte Ethernet-Verbindung stattfand. Der zweite Versuch wurde mit einer auf 10 Mbit/s gedrosselten Verbindung durchgeführt. Um der langsameren Verbindung gerecht zu werden, wurden auch nur 10 anstatt 100 GetItemValues-Nachrichten bei diesem Versuch versandt.

Test 1:

Netzwerkgeschwindigkeit: 100 Mbit/s Vollduplex

Anzahl der GetItemValues: 100

Versuchskonfiguration	Durchlauf 1	Durchlauf 2	Durchlauf 3	Mittelwert
Text, unkomprimiert	26,00	26,32	26,09	23,14
Text, GZIP	29,14	28,93	29,33	29,13
Text, ZLIB	29,96	29,40	30,18	29,85
Binär, unkomprimiert	3,02	2,72	2,88	2,87
Binär, GZIP	2,78	3,02	3,20	3,00
Binär, ZLIB	3,14	2,86	3,09	3,03

Abb. 7-2: Resultate des ungedrosselten Tests zur Übertragung eines Double-Kanal

Test 2:

Netzwerkgeschwindigkeit: 10 Mbit/s Vollduplex

Anzahl der GetItemValues: 10

Versuchskonfiguration	Durchlauf 1	Durchlauf 2	Durchlauf 3	Mittelwert
Text, unkomprimiert	32,89	27,25	31,84	30,66
Text, GZIP	19,71	20,89	16,94	19,18
Text, ZLIB	18,94	19,69	14,88	17,93
Binär, unkomprimiert	12,38	15,06	18,82	15,42
Binär, GZIP	13,77	12,34	12,73	12,95
Binär, ZLIB	12,66	13,03	12,23	12,64

Abb. 7-3: Resultate des gedrosselten Tests zur Übertragung eines Double-Kanal

Auswertung:

Im ersten Versuch ist sehr auffällig zu erkennen, dass eine Übertragung im binären Modus deutlich schneller ist als eine Übertragung der Nachricht in Textform. Dies liegt vor allem daran, dass jeder einzelne der Double-Werte in der Textnachricht als String übertragen wird und auf dem Empfängerhost wieder in eine Doublezahl umgewandelt werden muss. In der binären Nachricht hingegen wird der Double-Wert in seiner binären Repräsentation, wie sie auch im Speicher vorzufinden ist, übertragen. Diese binäre Repräsentation muss nun auf dem Empfängerhost nicht umgewandelt werden. Die einzige Operation, welche gegebenenfalls ausgeführt werden muss, ist die Konvertierung in eine andere Byte-Reihenfolge. So ist zu erkennen, dass im Versuch 1 die Übertragung im Binärmodus fast zehn Mal so schnell verläuft. Jedoch ist auch zu erkennen, dass die Übertragung der komprimierten Nachrichten bei dieser hohen Datenübertragungsrate von 100 Mbit/s sogar länger dauert. Dies ist dadurch zu erklären, dass die komprimierten Nachrichten zwar kleiner sind und somit bei dieser Datenübertragungsrate minimal schneller übertragen werden, als unkomprimierte Nachrichten, Dies aber nicht die Zeit aufwiegen kann, welche benötigt wird, um die Nachricht zu komprimieren und dekomprimieren.

Sieht man sich Versuch zwei an, so ist auch da zu erkennen, dass die Übertragung im Binärmodus schneller ist. Jedoch nicht sehr viel schneller. Dies legt daran, dass nun die Übertragung der Nachricht die meiste Zeit in Anspruch nimmt, da die Datenübertragungsrate des Netzwerks geringer und es somit deutlich langsamer ist. Das Parsen der Nachricht, welches im binären Übertragungsmodus deutlich schneller erfolgt, nimmt somit nur noch einen Bruchteil der Zeit ein im Gegensatz zur Übertragungszeig über das Netzwerk. Jedoch ist nun zu erkennen, dass komprimierte Nachrichten in kürzerer Zeit übertragen werden. Hierbei ist es nun so, dass die kleinere Nachricht so schnell übertragen wird, dass dies die Zeit, welche zum Komprimieren und Dekomprimieren benötigt wird, aufwiegt.

7.4 Fazit

Die vorangegangenen Tests haben einen kleinen Einblick in das Zeitverhalten der Nachrichtenübertragung zwischen dem EnCom Kern von iBEAM und einem EnCom Server geben können. So wurde festgestellt, dass bei kleinen Nachrichten, wie der Ping-

Nachricht die Übertragungsart und Komprimierung kaum eine Rolle spielt. Werden die Nachrichten jedoch größer und mit vielen Daten bestückt, welche im Textmodus geparst werden müssen, so wirkt sich die Übertragung im Binärmodus positiv aus. Unabhängig von Übertragungsmodus verbessert eine Komprimierung großer Nachrichten die Übertragungsgeschwindigkeit bei Netzwerken mit geringer Datenübertragungsrate, während sie bei Netzen mit einer hohen Datenübertragungsrate sogar eine negative Wirkung haben kann.

8 Konzepte der Oberflächenerweiterung

Nachdem nun der Kern von iBEAM erneuert wurde, befasst sich dieses Kapitel mit den Neuerungen an der Oberfläche von iBEAM und Untersuchungen, um diese nutzerfreundlich und informativ zu gestalten. Hierzu werden zuerst die Möglichkeiten und Anforderungen analysiert. Danach werden einige Konzepte vorgestellt. Dabei wird auch der „große Bruder“ des iPhones betrachtet - das iPad. Zu guter Letzt wird die neue Oberfläche vorgestellt und auf die Struktur eingegangen.

8.1 Möglichkeiten der Oberflächengestaltung auf mobilen Multi-Touch-Geräten des Hauses Apple

Die folgenden Unterkapitel gehen auf die Natural User Interfaces als zeitgenössische Mensch-Computer-Interaktionsform ein. Danach werden die natürlichen Interaktionsformen mit den Multi-Touch-Geräten von Apple untersucht.

8.1.1 Natural User Interfaces

Natural User Interfaces sind die neueste Generation von Schnittstellen zum Umgang zwischen Mensch und Computer.

Betrachtet man die Entwicklung von User Interfaces, so begann alles mit dem *Command Line Interface (CLI)* wie es zum Beispiel von MS-DOS genutzt wurde. Durch Eingabe von Kommandos oder Scripts[17] in Textform, konnte der Rechner gesteuert werden. Dies war umständlich, denn umso mehr Funktionen man nutzen wollte, umso mehr Befehle musste man im Kopf behalten.

Die Entwicklung des *Graphical User Interfaces (GUI)* brachte eine erheblich vereinfachte Bedienbarkeit für die breite Masse mit sich. Ab nun war die Interaktion über graphische Symbole möglich. Diese bieten viel mehr Möglichkeiten, Informationen zu übermitteln. So können Symbole und Bilder genutzt werden, um eine gewisse Intention hervorzuheben. Die Interaktion erfolgt dabei zumeist über eine Tastatur und ein Zeigegerät in Form einer Maus. Es gibt auch noch andere Eingabegeräte wie Touchpads oder Graphictablets[18]. Diese Bedienung ist eher indirekter Art, da der Nutzer Hilfsmittel benötigt, um mit dem Computer zu interagieren. Nichtsdestotrotz, ermöglichen diese neuen Eingabe- und

[17] Abfolge von Kommandos, welche zumeist in einer Datei abgelegt sind und abgearbeitet werden.

[18] Ein Zeigegerät, welches mittels eines speziellen Stiftes und Tabletts bedient wird. Die Position des Stiftes auf dem Tablett wird an den Computer übermittelt und ausgewertet.

Anzeigemöglichkeiten eine deutlich intuitivere, effizientere und nutzerfreundliche Bedienung, als die *Command Line Interfaces*.

Die *Natural User Interfaces* (*NUI*) stellen nun eine Erweiterung der *Graphical User Interfaces* dar. Dabei steht vor allem die Adaption der natürlichen Interaktionsmöglichkeit innerhalb der menschlichen Umgebung im Vordergrund. Hierzu eine Definition nach [Ric10] :

> ***"A Natural User Interface is a human-computer interface that models aspects of direct interactions between people and their natural environment." (Ein Natural User Interface stellt eine Mensch-Computer-Schnittstelle dar, welche Aspekte der direkten Interaktion zwischen Menschen und ihrer natürlichen Umgebung abbildet.)***

Die *Natural User Interfaces* erweitern somit vorrangig den Umfang der Interaktionsmöglichkeiten zur *GUI*, während die graphischen Oberflächen sich nicht großartig unterscheiden. Zur Interaktion wird sich der natürlichen Umgangsformen des Menschen bedient, wie:

- Berührung,
- Sicht,
- Stimme,
- Bewegung und
- Höhere Wahrnehmungen wie Ausdruck und Empfindung [Nat09]

Die Informationsübertragung soll dabei direkt und weitestgehend ohne Hilfsmittel, wie zum Beispiel eine Maus oder Keyboard erfolgen.

Das wohl populärste Beispiel für *Natural User Interfaces* sind (Multi-)Touch Geräte. Diese ermöglichen die direkte Steuerung per Finger und Gesten. So können Aktionen mittels „Fingertip" oder eines „Fingerstreichs" ausgeführt werden.

Ein weiteres Beispiel ist die „EyeToy Kamera", welche als Zubehör für die „Playstation 2" erhältlich ist. Sie ermöglichte eine Interaktion mit dem Spiel über Bewegungen des Spielers. [Son10]

Auch die Spielekonsole „Wii" bedient sich der Bewegung des Spielers als zentrales Steuerelement. So kann der Spieler zum Beispiel vom Wohnzimmer aus Golfen indem er nur die mit Bewegungssensoren ausgestattete Fernbedienung schwingt.

Eines haben jedoch alle natürlichen Benutzerschnittstellen gemeinsam: Sie sollen eine möglichst selbsterklärende und dem Nutzer gegenüber natürliche Interaktionsmöglichkeit mit Computern gewährleisten. Dies kann nicht nur eingesetzt werden, um die Produktivität zu steigern, sondern in vielen Fällen auch die Unterhaltsamkeit, wie deutlich am Einsatz von *NUI*s im Bereich der Spielekonsolen zu sehen ist.

8.1.2 Apple Multi-Touch-Geräte als Form eines Natural User Interfaces

Die Apple Multi-Touch-Geräte iPhone, iPod und iPad sind ein gutes Beispiel für Natural User Interfaces. Im Gegensatz zu den älteren Smartphone-Modellen[19], welche noch mit einer Tastatur ausgestattet waren, besitzt das iPhone nur noch vier Tasten: eine davon um in den Standby-Modus zu wechseln, zwei Tasten zur Regulierung der Lautstärke und eine Haupttaste, welche dazu genutzt wird, wieder in das Hauptmenü zurück zu kommen.

Ein deutliches Merkmal für die „Natürlichkeit" der Bedienoberfläche ist der Fakt, dass das iPhone ohne Bedienungsanleitung ausgeliefert wird. Es ist so intuitiv, dass fast jeder, der es zum ersten Mal in der Hand hält, sich sofort zurechtfindet. Dies macht einen großen Teil des Erfolges aus, da so auch Personen, welche mit der Bedienung von Computern und Handys Schwierigkeiten haben, sofort gefallen an der Bedienweise des iPhones finden können.

Die Steuerung des iPhones geschieht hierbei über Gesten[20]. So kann man mit einer „Wisch"-Bewegung umblättern oder aber auch einen Schieberegler bedienen, indem man mit dem Finger darauf klickt und diesen bewegt. Auch Gesten wie Schütteln oder Kippen des iPhones, wie es sehr oft bei Spielen verwendet wird, können zum Steuern verwendet werden.

8.1.3 Eigenschaften von "Apps"

Die *"Apps"* für das iPhone, den iPod oder das iPad sollten einigen Richtlinien folgen. Einerseits werden diese durch die Beschaffenheit der Geräte oder durch allgemeine Empfehlungen des Herstellers definiert. Dieses Unterkapitel bezieht sich dabei auf die iPhone Human Interface Guidelines und iPad Human Interface Guidelines [Ref10] .

8.1.3.1 Rahmenbedingungen der Hard- und Software

Die vorgegebene Hard- und Software des Endgerätes definiert die Anforderungen an Applikationen sehr stark und sollte bei der Entwicklung einer Anwendung unbedingt beachtet werden.

So ist zumindest für den iPod und das iPhone darauf zu achten, dass die Bildschirmgröße nur auf 3,5" beschränkt[21] ist. Das iPad besitzt ein großes Display von 9,7" und bietet somit auch mehr Platz für die Darstellung. Dabei kann man das Gerät drehen, so dass eine gute

[19] Mittlerweile besitzen viele Smartphone-Modelle auch Touch- bzw. Multi-Touch-Displays.
[20] Übersicht der Gesten siehe Anhang 2
[21] Übersicht der Technischen Daten siehe Anhang 7

Anwendung, wenn möglich, darauf reagieren sollte und sich im Hoch- oder Querformat ausrichten sollte.

Ein weiteres wichtiges Merkmal ist, dass *Apps* nur ein Fenster zur gleichen Zeit besitzen. Dies bedeutet, dass die Navigationsstruktur diesen Verhältnissen angepasst werden muss. Auch kann nur eine Applikation zur gleichen Zeit aktiv laufen. Ab dem *iOS* 4 ist das iPhone multitaskingfähig. Dies bedeutet, dass eine Anwendung im Hintergrund abgelegt werden kann und dort verweilt, bis sie beendet oder weitergeführt wird.

Da alle Apple Multi-Touch-Geräte für den mobilen Einsatz konzipiert wurden, sind sie auch strom- und platzsparend entworfen worden. Dies bedeutet auch, dass es begrenzte Ressourcen wie CPU oder Speicher gibt.

All dies sind durch die Hardware und das Betriebssystem festgelegte Rahmenbedingungen, welche bei dem Entwurf einer Anwendung und deren Oberfläche bedacht werden müssen.

8.1.3.2 Stile einer Anwendung

Es gibt ganz verschiedene Anwendungsgebiete für iPhone Applikationen. Diese haben auch unterschiedliche Ansprüche an ihr Auftreten und Wirken auf den Nutzer. Hierzu schlägt Apple drei Stile von Anwendungen vor:

- *Productivity Application*: Applikation, welche eher professioneller Natur sind, wichtige Aufgaben zu erledigen hat und somit zuverlässig und informativ auftreten muss. Dabei sind die Daten meist hierarchisch zugänglich und der Nutzer gelangt durch Auswahlprozesse zu den Informationen, welche er erhalten mag. Beispiel: E-Mail Applikation
- *Utility Application*: Applikationen, welche ein Minimum an Nutzereingaben benötigen um eine einfache Aufgabe zu erfüllen und auf die Anzeige von Informationen spezialisiert sind. Sie werden häufig zum Anzeigen eines Status benutzt und besitzen somit eine meist flache Struktur. Der Ablauf zeichnet sich dadurch aus, dass der Nutzer die Applikation öffnet, die Informationen abliest und gegebenenfalls Einstellungen ändert. Beispiel: Applikation, welche eine Wochenübersicht der Wettervorhersagen einer Stadt anzeigt.
- *Immersive Application*: Applikationen, welche meist als visuell ansprechende Fullscreen-Anwendungen laufen und Wert auf das Erlebnis des Nutzers legen. Hierzu gehören Spiele und nützliche Anwendungen, welche jedoch mehr Wert auf ein außergewöhnliches Auftreten legen. Somit zählen dazu Applikationen, welche für die breite Masse nutzbar sind und mit einem hohen Unterhaltungswert überzeugen. Meist bedienen sich diese Anwendungen nicht den vom Cocoa-Touch-Framework vorgegebenen Kontrollelementen, sondern warten mit eigens designten Oberflächen auf. Beispiel: Autorennspiele

8.1.3.3 Richtlinien der Gestaltung einer Anwendung

Jedoch gibt es auch Richtlinien für die Gestaltung einer iPhone App, welche unabhängig vom Anwendungsgebiet der Applikation sind und als allgemeingültig betrachtet werden können. Diese machen meist den Erfolg aus, da sie festlegen, wie gut der Nutzer mit der App arbeiten kann, ob sie intuitiv nutzbar ist und ob er Gefallen daran findet. Hierzu ist es wichtig, auf Gewohnheiten des Nutzers zu achten. Neue und für den Nutzer unlogische oder umständliche Bedienkonzepte können den Erfolg einer App mindern. Eine logisch strukturierte, ansprechende und intuitive Anwendung hingegen kann die Akzeptanz und die Zuneigung des Nutzers zu dieser Applikation steigern.

Auch hierzu schlägt Apple einige Richtlinien vor:

- Die in der Anwendung abgebildeten Vorgänge und Objekte sollen methaphorisch Dinge der realen Welt abbilden, um den Nutzer einen schnellen Einstieg in selbsterklärende Applikationen zu gewährleisten. Hierzu ist zu beachten, dass von dem *iOS* vorgegebene Kontrollelemente in ihrer Funktion beibehalten werden. Beispiel: Schieberegler, Ordner welche mit Dateien gefüllt werden
- Eine Aktion sollte eine sofortige Wirkung nach sich ziehen, so dass der Nutzer die Aktion sofort mit einem Resultat assoziieren kann.
- Nutzereingaben sollten so gering wie möglich gehalten werden, um somit den Nutzer von übermäßigen Eingaben mit der Tastatur zu entlasten und auch den Aufwand für die Fehlerprüfung dieser Eingaben zu minimieren. Dem Nutzer sollten stattdessen so oft wie möglich Optionen dargeboten werden, aus welchen er wählen kann.
- Der Nutzer sollte zu jeder Aktion, welche er tätigt eine Rückmeldung bekommen. Hierzu kann einerseits die sofortige Wirkung seiner Aktion zählen. Wenn diese jedoch eine gewisse Bearbeitungszeit benötigt, sollte der Nutzer darüber informiert werden. Dies kann mittels der Ausgabe einer Nachricht geschehen oder mittels eines *Activity Indicators*[22], welcher dem Nutzer anzeigt, dass im Hintergrund eine länger andauernde Aktion ausgeführt wird.
- Der Nutzer sollte jederzeit die Kontrolle über die Anwendung haben. Hierzu gehört zum Beispiel, dass der Nutzer einen länger andauernden Vorgang abbrechen kann.
- Eine Applikation sollte stets ein ansprechendes und der Funktion entsprechendes Auftreten haben. Eine professionelle Applikation, bei welcher vor allem die Informationen im Vordergrund stehen, sollte auch stets ein professionelles

[22] siehe Kontrollelemente Anhang 6

Auftreten mit wenig dekorativen Elementen, welche von der Kernaussage ablenken könnten, aufweisen. Bei Spielen hingegen steht oft der Unterhaltungsfaktor im Vordergrund und so ist es legitim, wenn diese knallige Farben, aufweisen um somit interessant und einladend auszusehen.

8.2 Entwicklung der neuen Oberfläche von iBEAM

Die folgenden Unterkapitel gehen auf den Entwicklungsprozess der neuen Oberfläche von iBEAM ein. Hierbei werden zuerst die Anforderungen und die darzustellenden Informationen analysiert. Danach werden Entwürfe und das fertige Produkt vorgestellt.

8.2.1 Anforderungen

Die Applikation iBEAM stellt eine Anwendung für professionelle Nutzer dar, welche gern Messdaten einer Serverapplikation visualisieren möchten. iBEAM soll somit den Stil einer *Productivity Application*[23] entsprechen. Hierzu gehört eine zuverlässig und informativ auftretende Oberfläche. Der Nutzer sollte sich sofort zurechtfinden und intuitiv durch die Programmstruktur geleitet werden. Die Oberfläche sollte in der Lage sein viele Informationen übersichtlich, strukturiert und schnell zugänglich anzuzeigen.

8.2.2 Darzustellende Informationen und Oberflächenentwürfe für das iPhone

Da iBEAM immer noch als Visualisierungsclient für messtechnische Daten dienen soll, werden sich auch die darzustellenden Informationen des neuen iBEAM nicht sonderlich von denen der Vorgängerversion unterscheiden.

Hierzu sollten weiterhin folgende Informationen und Eingabeelemente vorhanden sein:

- Eingabefelder für Port und IP Adresse des Servers
- Liste der auf dem Server verfügbaren Items. Es sollen jedoch alle verfügbaren Items und nicht nur unterstütze Items aufgelistet werden und auch graphisch unterschieden werden.
- Werte zu den abonnierten Kanälen
- Graphen zu den abonnierten Kanälen

Des Weiteren sollten nun noch folgende Informationen verfügbar sein:

- Allgemeine und Statistische Informationen zu Kanälen (z.B. Zeit der Erstellung, Produzent, Mittelwert der Werte, …)

---

[23] Siehe Kapitel 8.1.3.2

8.2.3 Struktur der Oberfläche

Das zu Beginn der Arbeit vorliegende iBEAM hatte eine Dreigliederung des Programms. So gab es eine Ansicht zum Verbinden mit dem Server und Anzeigen der Items. Dies war gleichzeitig die Startansicht. Über eine *TabBar* im unteren Bereich gelangte man zu den Ansichten für die Werte der Items und die Graphen. Hierbei war es so, dass in der Ansicht ausgewählt werden musste, welches Item betrachtet werden soll. Bei der Graphen-Ansicht hatte man dazu ein Pfeilsymbol im oberen Bildschirmbereich um durch die Liste der abonnierten Items durchzuschalten und bei der Werte-Ansicht gab es im rechten Bildschirmrand eine Leiste, welche alle abonnierten Items beinhaltete. Dies eignete sich jedoch nur für eine geringe Anzahl an Items, da diese Liste sonst zu überfüllt war.

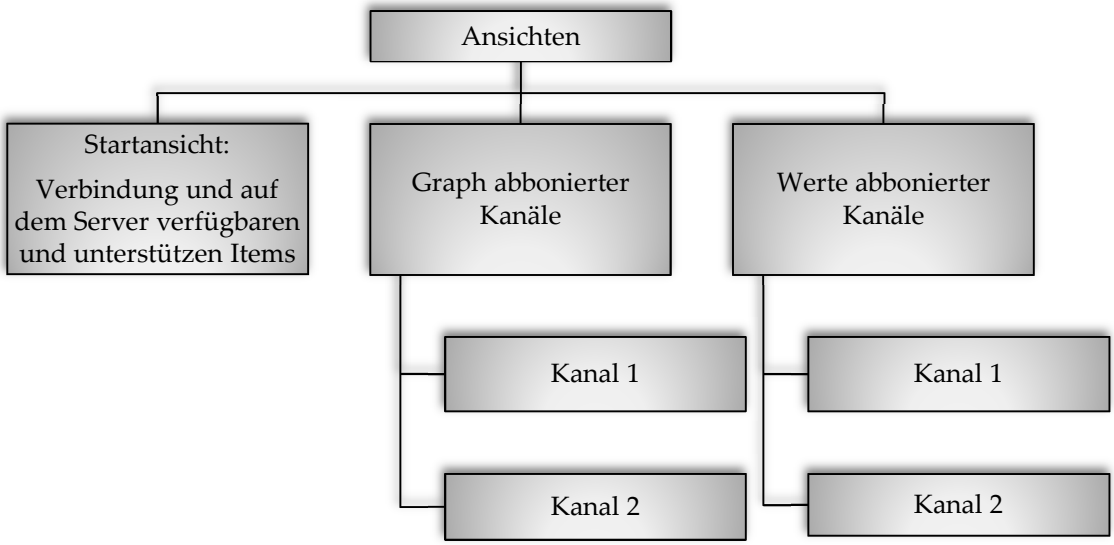


Abb. 8-1: Struktur der Graphischen Oberfläche des alten iBEAM (Quelle: Eigene Darstellung)

Die Struktur des modernisierte iBEAM soll etwas anders aufgebaut sein. So soll die Startansicht zwar wieder aus den Elementen zum Verbinden mit dem Server und der Liste aller verfügbaren Items auf dem Server bestehen, jedoch soll dies der Ausgangspunkt für alle weiteren Ansichten sein. Die Startansicht enthält also wieder eine Liste der Items des Servers. Bei einer Auswahlinteraktion mit einem Item soll man zu einer Detailansicht kommen, welche allgemeine und statistische Informationen des Items enthält.

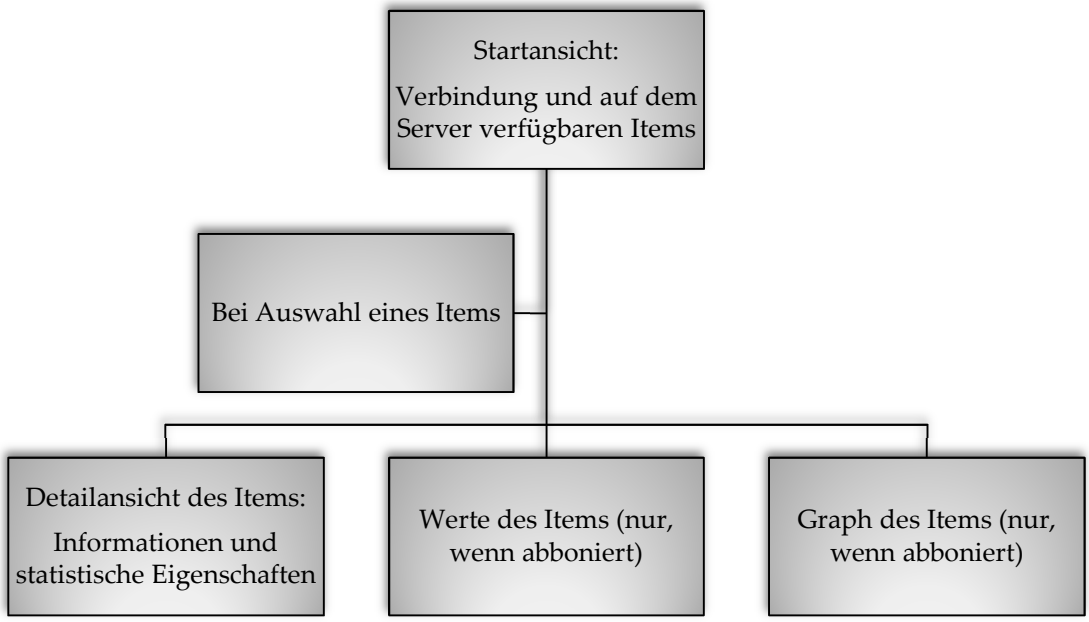


Abb. 8-2: Struktur der Graphischen Oberfläche des neuen iBEAM (Quelle: Eigene Darstellung)

Von der Detailansicht aus soll es möglich sein zwischen dieser Ansicht und den Ansichten des Graphen und der Werte dieses Items zu wechseln, vorausgesetzt, das Item ist abonniert. Auch soll man jederzeit wieder zurück zur Startansicht gelangen.

8.2.4 Entwürfe der Oberfläche für iPod Touch und iPhone

Die Startansicht des Programmes sollte wieder die Möglichkeit bieten, sich mittels IP Adresse und Port zu einem Server zu verbinden. Dies sollte auch im Laufe der Anwendung in dieser Ansicht sichtbar sein um dem Nutzer mitzuteilen, zu welchem Server er verbunden ist und gegebenenfalls die Verbindung zu beenden.

Auch soll beibehalten werden, dass auf dem Server verfügbare Items sofort sichtbar sind, nachdem sich verbunden wurde. Eine Tabelle ist hierfür am geeignetsten und somit soll auch dies von der alten Applikation übernommen werden. Unterschied zur alten Version soll hierbei nun sein, dass diese Items nach ihren Produzenten gruppiert werden. Dies ist auch in jBEAM der Fall und soll so auf iBEAM auch nachgebildet werden. Auch sollen alle verfügbaren Items angezeigt werden und nicht nur unterstützte Items. Jedoch muss dem Nutzer signalisiert werden, welche Items unterstützt sind und er somit abonnieren kann. Hierzu eignet sich eine farbliche Abhebung, wie zum Beispiel ausgrauen von Items, welche nicht abonniert werden können, da sie nicht von iBEAM unterstützt werden. Diese farbliche Abhebung kann auch bei der Unterscheidung abonnierter und nicht abonnierter Items vorgenommen werden. So eignet sich die Farbe Grün als Hintergrundfarbe für abonnierte Items. Items sollen nach wie vor bei einem *Tip* auf den Namen abonniert werden.

Um nun in die Detailansicht zu gelangen, wird sich eines *Detail Disclosure Buttons* bedient. Dies ist ein Button mit einem Pfeil, welcher oft für Tabellen Verwendet wird um in eine Detailansicht des Tabellenelements zu gelangen. Es soll auch für nicht abbonierbare Items möglich sein in die Detailansicht zu wechseln. Jedoch können dann keine Werte und kein Graph angezeigt werden.

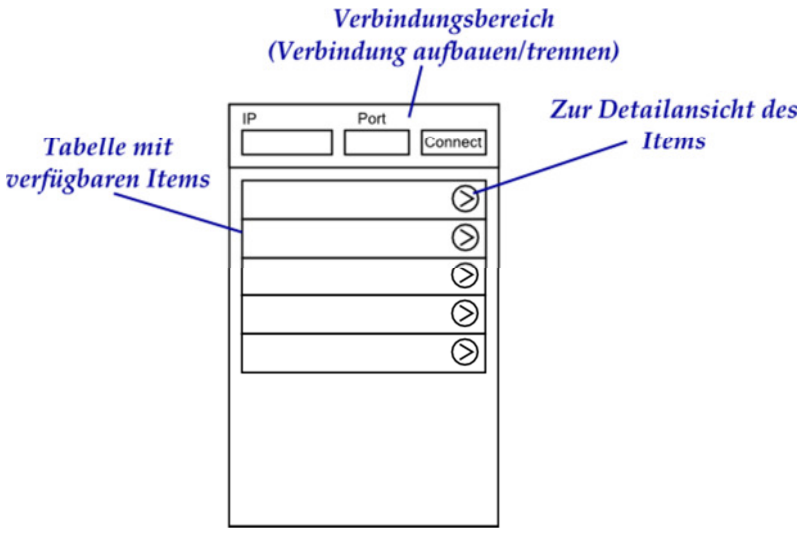


Abb. 8-3: Entwurf der Startansicht (Quelle: Eigene Darstellung)

Die Detailansicht soll nun allgemeine und statistische Informationen des Items anzeigen. Diese Informationen setzen sich aus einem Schlüssel und einem Wert zusammen. Wie zum Beispiel „Produzentenname“ und „DatGen“ als Wert. Da ein Item wieder mehrere dieser Schlüssel und Werte besitzt, bietet es sich an, auch diese wieder in einer Tabelle darzustellen. Im unteren Bereich der Ansicht, soll sich nun wiederum eine *TabBar* befinden, welche, im Falle eines abonnierten Items, zu der Werte- und der Graph-Ansicht leitet.

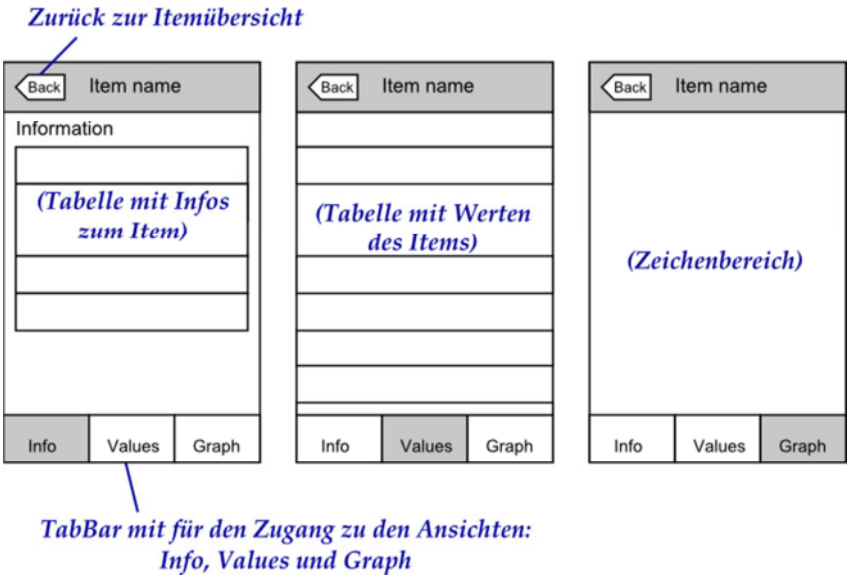


Abb. 8-4: Entwurf der Detailansichten (Quelle: Eigene Darstellung)

Diese Ansichten können dabei vom alten iBEAM übernommen werden. Somit werden die Werte wieder in einer Tabelle angezeigt und der Graph auf einem Zeichenbereich.

Diese drei Ansichten werden also Gleichwertig behandelt, aber stellen jeweils nur ein Item dar. Möchte man sich die Informationen, Werte oder den Graphen eines anderen Items anzeigen lassen, muss man in der Programmhierarchie wieder nach oben gehen. Dies ist mit einem „Zurück"-Knopf möglich und führt zurück zur Startansicht.

8.2.5 Umsetzung

Die Graphische Oberfläche wurde vollständig im Interface Builder, welcher dem Cocoa Framework beiliegt, erstellt. Hierzu wird für jede Ansicht, auch View genannt, eine eigene Interface-Builder-Datei angelegt. Diese enthält in der Regel ein *View*, auf welchem die graphischen Elemente wie Buttons oder Textfelder angeordnet werden. Außerdem können Klassen, welche in Xcode erstellt wurden, mit aufgenommen werden um somit im wahrsten Sinne des Wortes mit den graphischen Elementen verbunden zu werden. So können zum Beispiel Instanzvariablen, welche ein graphisches Element repräsentieren, mit dem entsprechenden Element per Drag and Drop verbunden werden oder Events können entsprechende Methoden zugeordnet werden, welche beim Eintreten abgearbeitet werden sollen.

Jede Ansicht erhält nun auch einen *View-Controller*. *View-Controller* sind dafür zuständig Events der Views abzufangen, neue Views zu laden oder Datenquellen für diese Bereitzustellen. So gibt es zum Beispiel *Table-View-Controller*, welche die Daten für Tabellen bereitstellen und zum Beispiel auch definieren, was beim *Tip* auf ein Element geschehen soll. *Navigation-Controller* können eine hierarchische Ordnung der Views verwalten.

Da die Tabellen mit den Werten eines Items und Informationen eines Items immer einen Schlüssel (z.B. Index des Wertes oder Name der Information) und einen dazugehörigen Wert anzeigen, bietet es sich an, dafür einen eigenen Zellentyp zu entwerfen. Dies bedeutet, dass eine Interface-Builder-Datei erstellt wird, welche kein *View* sondern eine *Table-View-Cell* als Hauptelement besitzt. Auf dieser Zelle können nun auch eigene Elemente wie Textfelder oder Graphiken angeordnet werden. Für die Schlüssel kann somit ein Label im linken Teil der Tabellenzelle angezeigt werden und im linken Teil der zugehörige Wert. Dieser kann zur optischen Hervorhebung auch größer dargestellt werden:

Somit ergibt sich nun diese geschachtelte Anordnung der graphischen Ansichten und der Interface-Builder-Dateien, welche diesen Ansichten angehören. Dabei ist in Klammern die Art des Hauptelements dieser Datei angegeben.

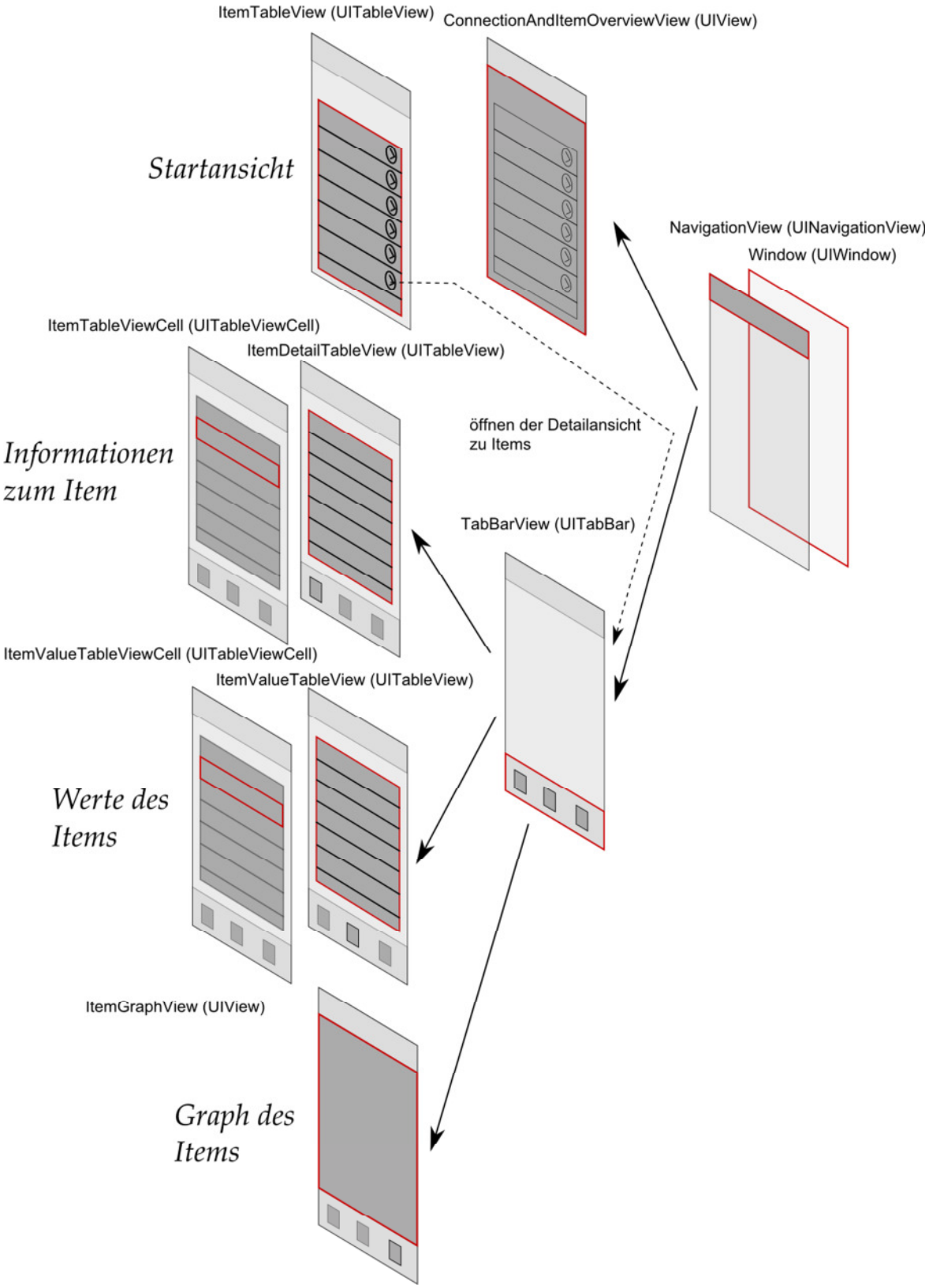


Abb. 8-5: Hierarchische Anordnung der Views am Beispiel der iPhone Oberfläche (Quelle: Eigene Darstellung)

8.2.6 Oberfläche des iPad

Die Oberfläche des iPad soll sich nicht grundlegend von denen für den iPod Touch und das iPhone unterscheiden, da es sich immer noch um die gleiche App handelt. So soll auch die Struktur und Hierarchie einheitlich sein. Jedoch sollte auch darauf eingegangen werden, dass das iPad eine größere Displayfläche besitzt und somit mehr Informationen auf einmal anzeigen kann.

Eine Stelle, an welcher mehr Informationen angezeigt werden könnten, wäre die Tabelle der verfügbaren Items. jBEAM besitzt die Funktion einer Übersichtsgraphik für Items. Diese ist ein kleines Bild, welches zum Beispiel den Graphen eines Kanals oder das erste Bild eines Videos darstellt. Da das EnCom Protokoll das Übertragen von Bildern unterstützt, kann diese Graphik vom Server erfragt werden.

Diese Übersichtsgraphik kann nun zusätzlich neben dem Namen des Items angezeigt werden. Für das Display des iPhones ist dies jedoch nicht geeignet, da es dafür zu klein ist.

8.3 Der fertige Prototyp

All die Überlegungen der vorrangegangenen Kapitel führen nun zum fertigen Prototyp. Im Folgenden werden die Funktionalität und das Aussehen dabei an der Oberfläche von iBEAM fürs iPhone aufgezeigt.

Der Startbildschirm der Applikation gleicht hierbei sehr dem Anblick des alten iBEAM.

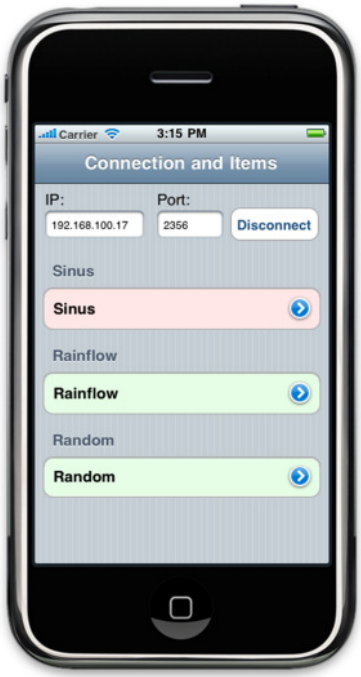


Abb. 8-6: Startansicht des neuen iBEAM (Quelle: Screenshot iPhone Simulator)

Im oberen Teil sind Textfelder und ein Button für den Aufbau und das Trennen der Verbindung zum Server vorhanden. Darunter findet sich eine Tabelle mit den auf dem Server verfügbaren Items. Diese sind nach ihren Produzenten gruppiert, so wie es zum Beispiel auch in jBEAM der Fall ist. Hierbei ist zu erwähnen, dass alle Items angezeigt werden und nicht nur unterstützte Items, wie es früher der Fall war. Zu nicht unterstützen Items zählen dabei Items, von denen keine Werte und kein Graph angezeigt wird. Dazu zählen zum Beispiel Video-Items, String Kanäle oder Mappenkanäle, welche Informationen zu Messungen enthalten können. Nicht unterstützte Items werden somit grau hinterlegt. Unterstütze Items sind rot hinterlegt, was signalisiert, dass sie nicht abonniert sind. Grün hinterlegte Kanäle hingegen sind abonniert. Der Abonnier- oder Abmeldevorgang wird dabei durch einen *Tip* auf den Namen des Items ausgelöst.

Ein *Tip* auf den Button, welcher neben jedem Item zu finden ist, leitet in eine Detailansicht des Items weiter. Von hier aus ist es möglich allgemeine und statistische Informationen zum Item anzusehen. Dies können Informationen zum Produzenten und zur Erzeugerzeit sein. Zu statistischen Informationen zählt zum Beispiel der Mittelwert, die Monotonie oder die Anzahl der Werte.

Wird das Item von iBEAM unterstützt und ist abonniert, so können sich zusätzlich noch die Werte und der Graph des Items angezeigt werden.

Abb. 8-7: Detailansichten zu einem Item im neuen iBEAM (Quelle: Screenshot iPhone Simulator)

In der Werteansicht findet sich dabei im rechten Displayrand eine Leiste, welche einen schnelleren Zugriff auf Werte in einem bestimmten Bereich erlaubt. So muss man nicht

immer durch die ganze Tabelle scrollen, sondern kann diese Leiste für Schnellzugriffe nutzen.

Die Visualisierungsroutinen für den Graphen wurden hierbei gänzlich vom alten iBEAM übernommen.

Die Oberfläche des iBEAM fürs iPad sieht ähnlich der Oberfläche für das iPhone aus. Bedingt durch das größere Display des iPads und die somit größere Fläche zur Anzeige von Informationen, wurde in der Startansicht eine Übersichtsgraphik zu den Items eingefügt. Diese Übersichtsgraphik wird von jBEAM, welches hierzu als Server diente, von Haus aus für jedes Item erzeugt und angezeigt. Mit einer neu eingeführten Nachricht namens GetItemImage kann diese Graphik nun angefragt werden.

Abb. 8-8: Startansicht von iBEAM auf dem iPad (Quelle: Screenshot iPhone Simulator)

9 Zusammenfassung

9.1 Erreichte Ziele

Alles in allem ist zu sagen, dass sich die nach außen hin sichtbaren Funktionalitäten von iBEAM nicht großartig geändert haben. Die graphische Oberfläche hat sich leicht verändert und ist logischer strukturiert. So kann der Nutzer nun in der Startansicht ein Item auswählen, welches ihn interessiert und findet dazu detaillierte Informationen und im Falle eines abonnierten Items auch die Werte und den Graphen. Möchte er sich dies von einem anderen Item ansehen, wechselt er zurück zur Startansicht und wählt ein anderes Item aus. Beim alten iBEAM hingegen mussten in der Graph- und Werteansicht jeweils aufs Neue das Item, welches den Nutzer interessiert, ausgewählt werden. Außerdem wurde die Oberfläche erstmals auf das iPad portiert.

Viel ausschlaggebender sind jedoch die Änderungen am Kern von iBEAM, welche nicht auf den ersten Blick von außen sichtbar sind. Hierzu gehört die vollständige Anpassung an die EnCom Protokollversion 3.0. Dabei wurde die Struktur der Java Implementierung des Protokolls übernommen um somit einen leicht zu wartenden und effizient arbeitenden Kern für iBEAM zu schaffen. So wurde der langsam arbeitende Textmodus für die Nachrichtenübertragung durch den viel schnelleren Binärmodus ersetzt. In Tests war dabei zu erkennen, dass dies bei großen Nachrichten mit vielen Werten eine fast zehn Mal so schnelle Übertragung gewährleistet.

Auch bildet die Neuimplementierung des EnCom Protokolls die Grundlage für spätere Erneuerungen und den Funktionsausbau von iBEAM, da es nun mehr Datentypen unterstützt werden. So werden folgende EnCom Datentypen schon unterstützt:

- DOUBLE, DOUBLE_1D
- FLOAT, FLOAT_1D
- SHORT, SHORT_1D
- INT, INT_1D
- LONG, LONG_1D
- STRING, STRING_1D
- OVERVIEW
- IMAGE
- MAP
- DATE
- ENUM

Dabei ist die Implementierung des EnCom Protokolls von der Applikation iBEAM unabhängig gehalten und könnte auch als Bibliothek in anderen Cocoa Programmen für den Mac oder iPhone genutzt werden.

9.2 Ausblick

Trotz der erfolgreichen Erneuerung des Kerns von iBEAM und einigen Anpassungen an der Oberfläche des Programms, gibt es doch noch viele Dinge, welche der Verbesserung bedürfen.

So sind noch nicht alle Enterprise Datentypen implementiert, welches jedoch durch die gute Struktur des EnCom Protokolls leicht nachzuholen ist. Auch kann das neue iBEAM weiterhin als Prototyp angesehen werden und sollte in dieser Form auch nicht zum Verkauf dargeboten werden. Jedoch kann iBEAM als Vorzeigemodel genutzt werden um zu zeigen, was mit dieser EnCom Implementierung möglich ist, um somit die Ideen Kunden anzuregen und eine Grundlage für kundenspezifische Erweiterungen zu schaffen. Den Möglichkeiten wären dabei kaum Grenzen gesetzt, da das EnCom Protokoll auch sehr generische Nachrichten anbietet, welche für jeglichen Zweck eingesetzt werden könnten. So könnten auch EnCom Nachrichten genutzt werden, um einem EnCom Server Steuerungsbefehle zukommen zu lassen. Außerdem könnte von einem iPhone oder iPad aus, Messungen auf einem entfernten Server gestartet oder beendet werden.

Weiterhin wäre es zum Beispiel möglich, die GPS Funktion des iPhones zu nutzen und in regenmäßigen Abständen die aktuelle Position an einen EnCom Server zu schicken, mit welchem wiederum andere Clients verbunden sein könnten um diese Daten zu empfangen und auch alle Updates dieser Daten zu erhalten.

Anhang 1 Größe der primitiven Datentypen in verschiedenen Programmiersprachen

(Quellen: [Meixner10] , [Bucanek09] S. 12)

Inhalt und Format	Vorzeichen-behaftet	Java	Objective-C	C Typedef	Größe im Speicher
zweiwertig (true, false)	-	boolean			1 Bit
	-	-	BOOL		1 Byte
16-bit Unicode	-	char	unichar	int16_t	2 Byte
8-bit Ganzzahl (Zweierkomplement)	Ja	byte	char	int8_t	1 Byte
	Nein	-	unsigned char	uint8_t	1 Byte
16-bit Ganzzahl (Zweierkomplement)	Ja	short	short int	int16_t	2 Byte
	Nein	-	unsigned short int	uint16_t	2 Byte
32-bit Ganzzahl (Zweierkomplement)	Ja	int	int	int32_t	4 Byte
	Nein	-	unsigned int	uint32_t	4 Byte
4-bit Ganzzahl (Zweierkomplement)	Ja	long	long long int	int64_t	8 Byte
	Nein	-	unsigned long long int	uint64_t	8 Byte
32-bit Gleitkommazahl (IEEE 754)	Ja	float	float	float_t	4 Byte
64-bit Gleitkommazahl (IEEE 754)	Ja	double	double	double_t	8 Byte

Abb. 9-1: Größe der primitiven Datentypen unter Java, Objective-C und als C Typedef

Anhang 2 Übersicht der Gesten zur Steuerung des iPhones

(Quelle: [Stäuble09] , S. 100 f.)

Geste	Übersetzung	Erläuterung
Tap	Tippen	Entspricht dem einfachen Klick.
Touch and Hold	Drücken und Halten	Entspricht dem Gedrückthalten der Maustaste auf einem Element.
Double Tap	Doppel Tippen	Entspricht einem Doppelklick.
Swipe	Schieben	Entspricht dem Schieben von Reglern.
Flick	Schubsen	Ein Wisch über das Display um Seiten umzublättern oder in Tabellen zu Scrollen.
Drag	Ziehen	Entspricht einem Gedrückthalten der Maustaste auf einem Element und Verschieben dessen.
Pinch	Strecken und Stauchen	Beide Finger werden auf die zu Vergrößernde Fläche gesetzt. Zusammenführen der Finger bewirkt eine Verkleinerung der Fläche und ein Auseinanderführen eine Vergrößerung.
Shake	Schütteln	Das Schütteln des Gerätes kann als Geste genutzt werden um eine Eingabe rückgängig zu machen.

Abb. 9-2: Übersicht der Gesten

Anhang 3 Übersicht der vom EnCom-Protokoll unterstützten Datentypen (Enterprise Datentypen)

Die folgenden Übersichten veranschaulichen die Datentypen, welche vom EnCom-Protokoll unterstützt werden und in Nachrichten übertragen werden können. (Quelle: [EnCom_Prot10] , S. 9ff)

Primitive Datentypen

Alle Primitiven Datentypen können als Einzelwerte und in Form von Arrays (bis zu drei Dimensionen) übertragen werden.

Datentyp	Wertebereich	Bezeichnung(en)
Boolean	true, false	BOOLEAN, BOOLEAN_1D, BOOLEAN_2D, BOOLEAN_3D
Byte	8 Bit Ganzzahl (Zweierkomplement, vorzeichenbehaftet)	BYTE, BYTE_1D, BYTE_2D, BYTE_3D
Short	16 Bit Ganzzahl (Zweierkomplement, vorzeichenbehaftet)	SHORT, SHORT_1D, SHORT_2D, SHORT_3D
Integer	32 Bit Ganzzahl (Zweierkomplement, vorzeichenbehaftet)	INTEGER, INTEGER_1D, INTEGER_2D, INTEGER_3D
Long	64 Bit Ganzzahl (Zweierkomplement, vorzeichenbehaftet)	LONG, LONG_1D, LONG_2D, LONG_3D
Float	32 Bit Gleitkommazahl (nach IEEE 754-1985)	FLOAT, FLOAT_1D, FLOAT_2D, FLOAT_3D
Double	64 Bit Gleitkommazahl (nach IEEE 754-1985)	DOUBLE, DOUBLE_1D, DOUBLE_2D, DOUBLE_3D

Abb. 9-3: Vom EnCom-Protokoll unterstütze, primitive Datentypen und deren Bezeichnung

Objektdatentypen

Datentyp	Beschreibung	Bezeichnung(en)
String	Zeichenkette in "" eingeschlossen.	STRING, STRING_1D, STRING_2D, STRING_3D
ChannelOverview	Eien Instanz der Klasse ChannelOverview.	OVERVIEW
Image	Image im PNG, JPEG oder GIF-Format	IMAGE, IMAGE_1D
File	Pfad zu einer Datei im String Format	FILE
URL	URL im Strring-Format	URL
Enum	Eine Enumeration (Enumerationswert, Enumerationsname)	ENUM
Class	Name einer Klasse im String-Format	CLASS
Map	Eine Map als Schlüssel-Wert-Zuordnung	MAP
Date	Millisekunden seit dem Referenzdatum 01.01.1970.	DATE
jbBinSerializable	Binär Serialisierte Objekte. Definiert durch Klassenname und der serialisierten Form als Bytearray.	JB_BIN_SERIALIZABLE

Abb. 9-4: Vom EnCom-Protokoll unterstütze Objektdatentypen und deren Bezeichnung

Anhang 4 Übersicht über die Nachrichten des EnCom-Protokolls

Für iBEAM relevante Nachrichten und Events

Name	Beschreibung
	Komandonachrichten
Hello	• erste Nachricht, welche versendet wird • Anfrage und Antwort enthalten Parameter, die Auskunft System geben
Ping	• Wird nach längerer Kommunikationspause von Server zum Client gesendet
GetAvailableItems	• Anfrage des Clients für eine Liste aller verfügbaren Items auf dem Server • Die Antwort enthält eine Liste aller verfügbaren Items mit Namen, IDs, Typen und Produzenten-Informationen
AddItemListener	• Der Client abonniert ein Item um über Änderungen an diesem informiert zu werden • Die Antwort enthält die Werte des Items und statistische Eigenschaften (wenn vorhanden)
RemoveItemListener	• Der Client meldet sich von einem Item ab und erhällt keine Änderungen an diesem mehr
GetItemInfo	• Der Client kann Informationen eines Items abrufen und ggf. die Daten des Items.
GetItemImage	• Neu eingeführte Nachricht um das Vorschaubild eines Items abzurufen
GetItemValues	• Client ruft einen Block der Daten eines Items ab.
GetItemImage	• Anfrage nach einer Übersichtsgrafik für das Item
Eventnachrichten	
ItemAvailable	• Der Server informiert über ein neues, verfügbares Item
ItemDataChanged	• Der Server informiert über die neuen Daten eines abonnierten Items
ItemNameChanged	• Der Server Informiert über den geänderten Namen eines Items
ItemReplaced	• Der Server informiert, dass ein Item durch ein anderes ersetzt wurde
ItemRevoked	• Der Server informiert über das Entfernen eines Items.
ItemPropertyChanged	• Der Server informiert über eine geänderte Eigenschaft des Items
ItemValuesAppended	• Der Server informiert über Werte, die an das Item angehängt wurden
ItemValuesReplaced	• Der Server informiert über Werte, welche von einem Item entfernt wurden
GetItemImage	• Neu eingeführte Nachricht zum Abrufen der Übersichtsgrafik eines Items

Abb. 9-5: Für iBEAM relevante EnCom Nachichten

Weitere Nachrichten und Events

Nachrichten, welche für das Programm iBEAM vorerst nicht relevant sind:

- ComponentNameChanged
- ComponentPropertyChanged
- GetComponentProperties
- InvokeComponentMethod
- Import
- Generic

Anhang 5 Quelltexte für den Test des DynamicDataBuffers

Nachfolgend sind die Quelltexte für einen Test des `DynamicDataBuffers` in Java und Objective-C zu finden. Sie weisen nach, dass die durch den `DynamicDataBuffer` erzeugte Bytefolge eines Floatarrays auf beiden Systemen identisch geschrieben und ausgelesen wird.

```
// Verbinden zum Server mittels Eingaben aus Textfeldern
Socket *sock = [Socket new];
[sock connect: [[IPEndPoint newWithIP: ip andPort: port]
                    autorelease]];

// Buffer anlegen und Array reinschreiben
DynamicDataBuffer *db = [[DynamicDataBuffer alloc] init];
float arr[] = {1.23456f, 9.8765f, 4.0f, 3.4567f};
Float1D *arrF = [Float1D newWithFloatArray: arr length: 4];

long len = [db writeFloatArray: arrF];

// Zurücksetzen des Positionszeigers um aufs Lesen
vorzubereiten
[db setPosition: 0];
// Bytearray für die Antwort
ByteT_1D *answerBytes = [ByteT_1D newWithLength: 2024];

// Schreiben in den Socket und auslesen der Antwort vom
Server
if ([sock isConnected]) {
   ByteT_1D *bytes = [db readBytes:len];

   [sock send: bytes.vals offset: 0 size: bytes.length];
   [sock receive: answerBytes.vals offset:0 size:
answerBytes.length];
}

// Leeren des Buffers um die empfangenen Bytes zu schreiben
// Empfangene Bytes Werden in Wrapperklasse gekapselt
[db clear];
[db writeBytes: answerBytes];

// Zurücksetzen des Positionszeigers um aufs Lesen
vorzubereiten
[db setPosition:0];

// auslesen des Floatarrays
Float1D *fArr = [db readFloatArray];
```

```
// Ausgeben des empfangenen Arrays
NSString *resText = @"";;
for(int i = 0; i<fArr.length; i++)
    resText = [resText stringByAppendingString: [@"  "
stringByAppendingFormat: @"%f", fArr.vals[i]]];

NSLog(resText);
// Konsolenausgabe: 1.234560  9.876500  4.000000  3.456700
```

Listing 9-1: Clientapplikation unter Objective-C, welche ein Floatarray überträgt und das selbe Arrays zurückgesendet bekommt

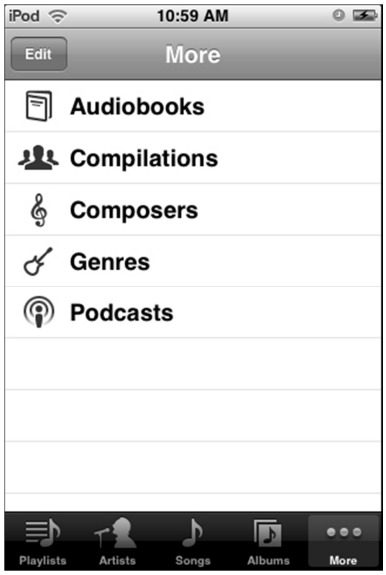
```
DynamicDataBuffer db =
DynamicDataBuffer.getDynamicDataBuffer();

ServerSocket server = null;
try {
    // Startet als Server mit Port 12345
    server = new ServerSocket();
    server.bind(new InetSocketAddress(12345));
    Socket client = null;
    client = server.accept();
    java.io.InputStream is = client.getInputStream();
    OutputStream os = client.getOutputStream();

    // Lesen der enpfangenen Bytes
    byte[] buffer = new byte[16384];
    int len = is.read(buffer);

    // bytes werden in den DynamicDataBuffer geschrieben
    db.write(buffer, 0, len);

    // Position zurückgesetzt um aufs Lesen vorzubereiten
    db.setPosition(0);

    // Auslesen und Ausgeben des Floatarrays
    float res[] = db.readFloatArray();
    for (int i = 0; i < res.length; i++)
          System.out.print(res[i] + " ");
    // Konsolenausgabe: 1.23456 9.8765 4.0 3.4567

    // Leeren des Buffers um das Floatarray wieder
hineinzuschreiben
    db.clear();
    len = (int) db.writeFloatArray(res);

    // Position zurückgesetzt um aufs Lesen vorzubereiten
    db.setPosition(0);
```

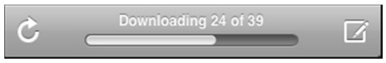
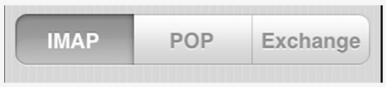
```
    // Zurücksenden des Empfangenen Arrays
    byte send[] = db.readBytes(len);
    os.write(send);
}
```

Listing 9-2: Serverapplikation unter Java, welches ein Floatarray empfängt, ausgibt und wieder zurücksendet

Anhang 6 Übersicht der Kontrollelemente für Graphische Oberflächen auf dem iPhone und iPad

Auswahl einiger Kontrollelemente auf den iPhone und iPad (Quelle: [AppleRef10] , iPhone Human Interface Guidelines)

Bezeichnung	Beispiel	Eigenschaften
NavigationBar		• Eine Leiste zur Navigation • Optionen um Navigationsebenen zurückzuspringen, aktuelle Ansichten zu editieren oder den aktuellen Standpunkt aus Navigationssicht anzuzeigen • Meist im oberen Bereich der Ansicht
TabBar		• Leiste mit Tabs zum Wechseln von Ansichten • Meist im unteren Bereich der Ansicht
ToolBar		• Leiste mit Feldern für Optionen • Meist im unteren Bereich der Ansicht
TableView		• Stellt eine Tabelle dar • Die Zellen der Tabelle können individuell gestaltet werden. So können Bilder, Texte und Steuerelemente wie Buttons, Schieberegler oder Switches eingebaut werden. •
Switch		• Schalter für Ja-Nein-Optionen
Text View		• Bereich für Textanzeige und –eingabe • Zum Editieren des Bereiches wird eine Teastatur eingeblendet

WebView		• Bereiche in denen Web Content angezeigt werden kann
Label		• Beschriftung in Textform
Date and Time Pickers		• Element zum Wählen von Datums- und Uhrzeitangaben.
Page Indicator		• Meist im unteren oder oberen Teil der Ansicht befindliche Leiste • Punkte stellen die einzelnen Seiten dar • Hervorgehobener Punkt ist aktuelle Seite
Picker		• Vergleichbar mit einer Combobox zur Auswahl aus mehreren Elementen
Progress View		• Eine Leiste, welche Fortschritte anzeigt.
Segmented Controll		• Ein Segmented Controll besteht aus mehreren Schaltflächen, wobei dabei meistens jede für eine Option steht, welche Grundlage für folgende Einstellungen ist.

Anhang 7 Technische Daten der Apple Multi-Touch-Geräte der aktuellen Generation und der Vorgängermodelle

Quellen: [Apple10] , [Wik10]

iPhone

Model	**iPhone 3GS**	**iPhone 4**
		
OS bei Auslieferung	iPhone OS 3	iOS 4
Display	Multi Touch Display 480 x 320 Pixel	Retina Multi Touch Display 960 x 640 Pixel
Speicher	8, 16, 32 GB	16, 32 GB
Prozessor	833 MHz (getaktet mit 600 MHz) ARM Cortex-A8	ARM Cortex-A8 Apple A4
Arbeitsspeicher	256 MB DRAM	512 MB DRAM
Mobilfunkt und Wi-Fi	• UMTS/HSDPA (850, 1900, 2100 MHz) • GSM/EDGE (850, 900, 1800, 1900 MHz) • 802.11b/g Wi-Fi • Bluetooth 2.1 + EDR wireless technology	• UMTS/HSDPA/HSUPA (850, 900, 1900, 2100 MHz) • GSM/EDGE (850, 900, 1800, 1900 MHz) • 802.11b/g/n Wi-Fi (802.11n nur mit 2,4 GHz) • Bluetooth 2.1 + EDR

iPod

Model	**iPod Touch 3. Generation** 	**iPod Touch 4. Generation** 
OS bei Auslieferung	iPhone OS 3.1.1	iOS 4.1
Display	Multi Touch Display 480 x 320 Pixel	Retina Multi Touch Display 960 x 640 Pixel
Speicher	32, 64 GB	8, 32, 64 GB
Prozessor	833 MHz (getaktet mit 600 MHz) ARM Cortex-A8	ARM Cortex-A8 Apple A4
Arbeitsspeicher	256 MB RAM	256 MB
Mobilfunkt und Wi-Fi	• 802.11b/g Wi-Fi • Bluetooth 2.1 + EDR wireless technology	• Wireless3 • 802.11b/g/n Wi-Fi (802.11n 2.4GHz only) • Bluetooth® 2.1 + EDR • Maps-location based service4 • Nike + iPod support built in •

iPad

Model	**iPad** 
OS bei Auslieferung	iPhone OS 3.2
Display	LED-backlit Multi Touch Display 1024x768 Pixel
Speicher	16, 32, 64 GB
Prozessor	ARM Cortex-A8 Apple A4
Arbeitsspeicher	256 MB DRAM
Mobilfunkt und Wi-Fi	Wi-Fi model • Wi-Fi (802.11a/b/g/n) • Bluetooth 2.1 + EDR technology Wi-Fi + 3G model • UMTS/HSDPA (850, 1900, 2100 MHz) • GSM/EDGE (850, 900, 1800, 1900 MHz) • Data only2 • Wi-Fi (802.11a/b/g/n) • Bluetooth 2.1 + EDR technology

Anhang 8 Cross Compiler Beispiel mit XMLVM

„Hello World“ mit dem Cross Compiler XVML [XMLVM10] von Java nach Objective-C konvertiert.

```
public class HelloWorld
{
    public static void main(String[] args) {
          System.out.println("Hello World!");
    }
}
```

Listing 9-3: “Hello World” in Java

```
<?xml version="1.0" encoding="UTF-8"?>
<vm:xmlvm xmlns:vm="http://xmlvm.org"
xmlns:dex="http://xmlvm.org/dex">
  <vm:class name="HelloWorld" package="" extends="java.lang.Object"
isPublic="true">
    <vm:method name="&lt;init&gt;" isPublic="true">
      <vm:signature>
        <vm:return type="void" />
      </vm:signature>
      <dex:code register-size="1">
        <dex:var name="this" register="0" type="HelloWorld" />
        <vm:source-position file="HelloWorld.java" line="2" />
        <dex:invoke-direct class-type="java.lang.Object"
method="&lt;init&gt;" register="0">
          <dex:parameters>
            <dex:return type="void" />
          </dex:parameters>
        </dex:invoke-direct>
        <dex:return-void />
      </dex:code>
    </vm:method>
    <vm:method name="main" isStatic="true" isPublic="true">
      <vm:signature>
        <vm:parameter type="java.lang.String[]" />
        <vm:return type="void" />
      </vm:signature>
      <dex:code register-size="3">
        <dex:var name="var-register-2" register="2"
type="java.lang.String[]" />
        <vm:source-position file="HelloWorld.java" line="9" />
        <dex:sget-object kind="field" class-type="java.lang.System"
member-type="java.io.PrintStream" member-name="out" vx="0" vx-
type="java.io.PrintStream" />
        <dex:const-string kind="string" value="Hello World!" vx="1"
vx-type="java.lang.String" />
        <dex:invoke-virtual class-type="java.io.PrintStream"
```

```
method="println" register="0">
          <dex:parameters>
            <dex:parameter type="java.lang.String" register="1" />
            <dex:return type="void" />
          </dex:parameters>
        </dex:invoke-virtual>
        <vm:source-position file="HelloWorld.java" line="11" />
        <dex:return-void />
      </dex:code>
    </vm:method>
  </vm:class>
</vm:xmlvm>
```

Listing 9-4: "Hello World" als XMLVM Datei

```
+ (void) main___java_lang_String_ARRAYTYPE :(XMLVMArray*)n1
{
    XMLVMElem _r0;
    XMLVMElem _r1;
    XMLVMElem _r2;
    _r2.o = n1;
    _r0.o = JAVA_NULL;
    _r1.o = JAVA_NULL;
    [_r2.o retain];
    _r0.o = [java_lang_System _GET_out];
    [_r0.o retain];
    _r1.o = @"Hello World!";
    [((java_io_PrintStream*) _r0.o)
println___java_lang_String:_r1.o];
    [_r0.o release];
    [_r1.o release];
    [_r2.o release];
    return;
}
```

Listing 9-5: Generiertes Objective-C "Hello World"

Quellen

[jBEAM10] AMS Gesellschaft für angewandte Meß- und Systemtechnik mbH. 2010. *jBEAM und seine Komponenten.* Flöha : s.n., 2010.

[AMS10] AMS Gesellschaft für angewandte Mess- und Systemtechnik mbH. 2010. AMS Gesellschaft für angewandte Mess- und Systemtechnik mbH. [Online] 2010. [Zitat vom: 30. Juni 2010.] http://www.jbeam.de.

[EnCom_Prot10] AMS Gesellschaft für angewandte Mess und Systemtechnik mbH. 2010. EnCom Enterprise Communication. *Protocol Documentation.* Flöha : s.n., 2010.

[EnCom_Arch10] AMS Gesellschaft für angewandte Mess- und Systemtechnik mbH. 2010. EnCom Enterprise Communication. *Architecture Documentation.* Flöha : s.n., 2010.

[EnCom_Prä10] —. 2010. EnCom version 2. *Presentation of the EnCom architecture version 2.* [Präsentation]. 2010.

[Apple10] Apple Inc. 2010. Apple. [Online] 2010. [Zitat vom: 12. August 2010.] www.apple.com.

[AppleRef10] —. 2010. iOS Reference Library. [Online] 2010. http://developer.apple.com/iphone/library/navigation/index.html.

[Bucanek09] Bucanek, James. 2009. *Learn Objective-C for Java Developers.* New York : Springer-Verlag, 2009.

[Fritzler10] Fritzler, Tatjana. 2010. Konzeption und Entwicklung eines Visualisierungsclients für iPhone/iPod touch. *Diplomarbeit.* Flöha : s.n., 2010.

[Intel10] Intel Coorperation. 2004. Endianness White Paper. [Online] 2004. [Zitat vom: 07. September 2010.] http://www.intel.com/design/intarch/papers/endian.pdf.

[Koc09] Kochan, Stephen G. 2009. *Objective-C 2.0 Anwendungen Entwickeln für Mac und iPhone.* München : Addison-Wesley Verlag, 2009.

[Meixner10] Meixner, Prof. Dr. Gerhard. Datentypen: Einfache Datentypen. [Online] [Zitat vom: 22. September 2010.] http://www.hs-augsburg.de/~meixner/prog/skript/datentypen/EinfacheDatentypen.html.

[Monson10] Monson-Haefel, Richard. 2010. Proposed Definition of Natural User Interface (NUI). *The iPad Entrepreneur.* [Online] Januar 15, 2010. [Cited: September 15, 2010.] http://theclevermonkey.blogspot.com/2010/01/proposed-definition-of-natural-user.html.

[NUIG09] Natural User Interface Group. 2009. NUI Group Community FAQs. [Online] 27. Juli 2009. [Zitat vom: 15. September 2010.] http://nuigroup.com/faq/.

[Schubert10] Schubert, Prof. Dr.-Ing. Wilfried. 2010. Fach Softwaretechnik. *Lehrmaterial.* 2010.

[Sony10] Sony Computer Entertainment Europe. EyeToy USB-Kamera. [Online] [Zitat vom: 15. September 2010.] http://de.playstation.com/ps2/peripherals/detail/item51693/EyeToy-USB-Kamera/.

[Stäuble09] Stäuble, Markus. 2009. *Programmieren fürs iPhone - Einstieg in die Anwendungsentwicklung mit dem iPhone SDK 3.* Heidelberg : dpunkt.verlag, 2009.

[IEEE85] The Institute of Electrical and Electronics Engineers, Inc. 1985. IEEE Standard for Binary Floating-Point Arithmetic. [Online] 1985. [Zitat vom: 17. September 2010.] http://754r.ucbtest.org/standards/754.pdf.

[Wik10] Wikimedia Foundation Inc. 2010. Wikipedia. [Online] 2010. [Zitat vom: 10. September 2010 .] http://de.wikipedia.org/.

[XMLVM10] XMLVM. [Online] [Zitat vom: 14. Juli 2010.] http://www.xmlvm.org/.

Verwendete Hilfsmittel

Entwicklungsumgebung:

- Xcode und Interface Builder, Version 3.2.3, 64 bit
- Instruments, Version 2.7

Testgeräte:

- iPod Touch, 1. Generation, 8 GB
- iPad

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

……