Cwikowski, Henryk

Entwicklung einer erweiterbaren Softwarearchitektur zur Barcodeerfassung

eingereicht als

Bachelorarbeit

an der

Fachbereich Mathematik/Physik/Informatik

Mittweida, 2009

Erstprüfer: Prof. Dr. Joachim Geiler
Zweitprüfer: Dipl.-Ing. Frank Thomas

Bibliographische Beschreibung:

Cwikowski, Henryk: Entwicklung einer erweiterbaren Softwarearchitektur zur Barcodeerfassung - 2009. - 54 S. Mittweida, Hochschule Mittweida, Fachbereich MPI, Bachelorarbeit, 2009

Referat:

Diese Bachelorarbeit beschreibt das Vorgehen des Entwurfs einer erweiterbaren Softwarearchitektur zur Barcodeerfassung. Hintergrund der Arbeit ist die Idee eines Systems, um verschiedene Module, wie zum Beispiel Inventurerfassung oder Wareneingangserfassung, in ein bestehendes System zu integrieren, ohne eine komplette Neuentwicklung durchzuführen. Häufig treten diese Änderungen auf, wenn das Programm für einen Kunden angepasst oder eine neue Hardwaretechnologie (z.B. RFID) eingeführt wird. In dem Fall soll nur das entsprechende Modul bzw. der Gerätetreiber ausgetauscht werden, ohne auf den bestehenden Programmcode zuzugreifen.
Im Rahmen dieser Bachelorarbeit wird die Aufgabe eines Softwarearchitekten übernommen. Dabei wird versucht, eine erweiterbare Softwarearchitektur zur Barcodeerfassung zu entwickeln. Das Konzept wird bestehende Technologien sowie verschiedene Gerätetypen berücksichtigen und ein leichteres Hinzufügen von Modulen ermöglichen. Im Abschluss soll ein entsprechender Prototyp entwickelt werden, welcher die Flexibilität bestätigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Nahezu jeder industrielle Betrieb nutzt zur Identifizierung der Produkte eindeutige Nummern oder Bezeichnungen. Diese liegen in Regel in codierter Form vor, um von elektronischen Geräten gelesen zu werden. Auch wenn eine Investition in diese Geräte einen zusätzlichen Kostenfaktor darstellen, so erzielen sie eine hohe Zeitersparnis von ca. 55% bei Inventuren und senken die Fehlerquote durch Schreibfehler auf nahezu 0% [01]. Durch die elektronische Erfassung, die meist schon bei den Zulieferern beginnt, ist eine lückenlose Rückverfolgung der Ware, bzw. ein Überprüfen des aktuellen Statuses möglich. Das bietet die Möglichkeit, effektiver auf Produktengpässe, Nachbestellungen oder Preisänderungen einzugehen.
Da zwischen den MDE-Geräten große Preisunterschiede bestehen, sollte genau abgewägt werden, inwiefern sich eine höhere Invenstition rechnet. Kleine

Geräte kosten 300€ bis 500€, während sich der Preis modernerer Geräte mit neuen Technologien auf ca. 1000€ bis 2000€ beläuft [02],[03],[04]. Durch die Investition in diese neuen Technologien erhält der Betrieb sehr robuste und zukunftssichere Geräte. Sie bieten die verschiedensten Möglichkeiten der Datenübertragung von USB über Bluetooth / Infarot bishin zu LAN / WLAN .

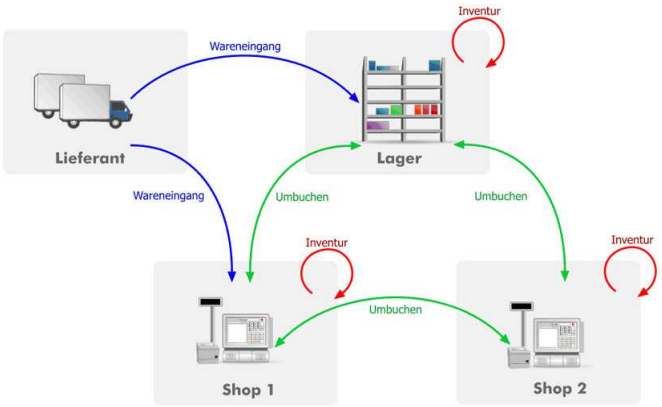

Abbildung 1.1: Barcodeerfassung innerhalb eines Betriebes

Ein wichtiger Faktor, der bei einer Anschaffung von MDE-Geräten berücksichtigt werden muss, ist die Form der Datenerfassung. Zur Zeit existieren zwei verschiedene Technologien auf dem Markt: Barcode-Scanner (Laser oder CCD) und RFID-Scanner (Funk)[05]. Traditionell werden Barcodescanner genutzt, da sich diese in der Vergangenheit bewährt haben, während RFID eine neuere Technologie darstellt. Bei dieser werden die Daten per Funk von einem Chip übertragen, der in das Produkt integriert ist. Das hat den Vorteil, dass die Etiketten kontaktlos und daher viel schneller erfasst werden können, als es mit einem Barcodescanner der Fall wäre.
Erfolgte eine Festlegung auf eine Technologie bzw. ein Gerät, sollte über weitere Anwendungsmöglichkeiten dieser Geräte nachgedacht werden, da der Kostenfaktor zu hoch ist, um auf den MDE-Geräten nur eine einzige Inventursoftware zu installieren und die Geräte für nichts Weiteres zu nutzen. Dabei sollten auch Möglichkeiten in Betracht gezogen werden, welche nicht zwangsläufig in den Bereich der Barcodeerfassung fallen, wie zum Beispiel Arbeitszeiterfassung

oder die Verwendung des MDE-Gerätes als Handy und Organizer. Allerdings ist es nicht einfach, sich diese Ideen im Voraus zu überlegen, ohne über einen längeren Zeitraum mit diesen Geräten gearbeitet zu haben, da allein durch eine Produktbeschreibung nicht das ganze Potenzial bekannt wird. Das würde dazu führen, dass beispielsweise nur eine einfache Inventursoftware entwickelt wird, ohne auf ein grundlegendes dynamisches Konzept zu achten, da dies für ein so kleines Projekt nicht notwendig ist. Nach einem halben Jahr wird festgestellt, dass sich diese Geräte auch zur Erfassung des Wareneinganges anbieten würden. Die Geräte werden zur Erweiterung zurück zur Entwicklerfirma geschickt und die Programmierer beginnen von Neuem, sich in die hardwarespezifischen Einstellungen des Gerätes einzuarbeiten. Das ist notwendig, da jeder Gerätehersteller eigene Treiber entwickelt und dafür verschiedene Schnittstellen zur Verfügung stellt. Die Dokumentationen dafür können sich auf mehrere hundert Seiten belaufen und sind von Hersteller zu Hersteller sehr verschieden. Der erneute Arbeitsaufwand stellt einen zusätzlichen Kostenfaktor für das Projekt dar und könnte unter bestimmten Voraussetzungen vermieden werden, z.B wenn bei der ersten Entwicklung auf eine dynamische und erweiterbare Softwarearchitektur geachtet wird.
Die Architekturen bilden das Grundgerüst eines jeden Programms und müssen daher tiefgründig geplant werden. Dabei reicht es nicht, eine triviale Architektur zu entwerfen, von der man annimmt, dass sie sich später bewähren wird. Es ist sehr wichtig, dass alle Anforderungen, Einflussfaktoren und Qualitätsmerkmale schon in der Architektur berücksichtigt werden. Je nach Größe des Projektteams ist das die Aufgabe eines Softwarearchitekten. Er verfügt über das notwendige Wissen der benötigten Technologien, Architekturstile und Entwicklungsumgebungen, besitzt aber auch gute Programmierkenntnisse, da er teilweise selbst in das Projekt mit eingreifen muss. Nachdem er den ersten Architekturentwurf erstellt hat, übernimmt er die Entwicklung eines Prototypen, um eventuelle Probleme in der Architektur noch vor der Implementierungsphase zu beheben.

1.2 Zielsetzung

Im Rahmen dieser Bachelorarbeit wird die Aufgabe eines Softwarearchitekten übernommen. Dabei wird versucht, eine erweiterbare Softwarearchitektur zur Barcodeerfassung zu entwickeln. Die Architektur soll dabei bestehende Technologien sowie verschiedene Gerätetypen berücksichtigen und ein leichteres Hinzufügen von zusätzlichen Modulen ermöglichen. Im Abschluss soll ein entsprechender Prototyp entwickelt werden, welcher die Flexibilität bestätigt.

1.3 Struktur der Arbeit

Die vorliegende Bachelorarbeit beschreibt das Vorgehen zur Erstellung einer erweiterbaren Softwarearchitektur. Im ersten Schritt werden einige theoretische Grundlagen beschrieben. Dabei wird auf den Aufbau und die Funktionsweise von Barcodes eingegangen sowie die verschiedenen Erfassungsmöglichkeiten von Scannern und RFID erläutert.
Der Hauptteil widmet sich dem eigentlichen Thema - der Softwarearchitektur. Zuerst wird der Leser über die allgemeine Vorgehensweise zur Architekturerstellung sowie über die Funktionsweise bestehender Systeme informiert. Des weiteren werden alle Anforderungen und Einflussfaktoren bestimmt, die für die Erstellung der Architektur notwendig sind. Aus diesen gewonnenen Daten können bereits Schnittstellen definiert und ein Architekturentwurf abgeleitet werden.
Abschließend wird eine Analyse und Bewertung der entwickelten Architektur durchgeführt, damit mögliche Risiken ausgeschlossen werden. Einen konkreten Beweis für die Flexibilität wird ein Prototyp liefern, der gleichzeitig zukünftigen Entwicklern einen Einblick in die Vorgehensweise ermöglicht.

2
Barcodeerfassung

2.1 Barcodes

Als Strichcode, Balkencode oder Barcode wird eine optoelektronisch lesbare Schrift bezeichnet, die aus verschieden breiten, parallelen Strichen und Lücken besteht. Die Daten ergeben sich aus der binären Kodierung der Anordnung von Lücken und Strichen. Oft steht in der Klartextzeile direkt unter dem Barcode der Dateninhalt zusätzlich in für Menschen lesbarer Schrift[06]. So kann der Anwender bei etwaigen Leseproblemen die Information manuell auswerten.
Damit eine hohe Erstleserate der Barcodes erzielt werden kann, müssen gewisse Richtlinien eingehalten werden. Empfohlen wird ein Farbkontrast zwischen Lücken und Strichen von durchschnittlich mindestens 35%. Generell gilt, dass für Lücken helle Farben, wie gelb, orange oder auch rot gut geeignet sind. Dementsprechend sollten für die Striche kräftige Farben wie grün, blau oder schwarz gewählt werden. Nur wenn die Verpackungen die Vorgaben erfüllen,

wird das jeweilige Produkt in das Sortiment aufgenommen. Eine vorherige Abstimmung bezüglich Farbgebung und Positionierung des Barcodes auf der Verpackung kann somit den Vertrieb und dadurch natürlich den Umsatz eines Produktes in entscheidender Weise beeinflussen[07].
In der nachfolgenden Liste ist zu sehen, welche Barcodes hauptsächlich in der Wirtschaft genutzt werden.

Handel(POS):	EAN 8, EAN 13, EAN 128 UPC-A, UPC-E Code 39
Industrie:	Code 2/5 Interleaved 2D Codes (Data Matrix)
Elektronik:	Code 39, Code 128
Logistik, Transport und Paketdienste:	EAN 13, EAN 128 Code 39, Code 128 PDF 417, Data Matrix
Chemische Industrie, Gesundheitswesen:	EAN 13, EAN 128 Code 39, Code 128 RSS-Code, Data Matrix
Apotheken:	Code 39

2.2 Codetypen

2.2.1 EAN-Code

Die EAN steht für International Article Number (früher European Article Number) und ist eine Produktkennzeichnung für Handelsartikel. Diese Nummer besteht aus 8 oder 13 Ziffern und ist wie folgt aufgebaut :

- 1.-2. Ziffer: identifiziert das Land des Herstellers
- 3.-7. Ziffer: Herstellernummer
- 8.-12. Ziffer: Artikelnummer des Herstellers
- 13. Ziffer: Prüfziffer des EAN 13 Codes

Der EAN-Code ist numerischer Code und besitzt eine hohe Informationsdichte(10 Billionen Zahlen). Das Größenverhältnis (Höhe x Breite) ist festgelegt, d.h. der Barcode kann nicht beliebig in Höhe und Breite verändert werden [08]. Eine neue Variante der EAN ist die GTIN (Global Trade Item Number). Sie besteht ebenfalls auf 13 Ziffern, hat aber einen flexiblen Aufbau. Die Herstellernummer kann 7- bis 9-stellig sein. In Abhängigkeit zur Länge dieser Nummer können für die individuelle Artikelnummer drei, vier oder fünf weitere Ziffern angehängt werden. Damit variiert die freie Nummernkapazität zwischen 1.000 bis 100.000 GTIN-Artikelnummern .

2.2.2 Code 2/5 Interleaved

Der Code 2/5 Interleaved ist nur ein Beispiel aus der Code2/5-Familie, die alle numerisch sind. Der Code 2/5 Interleaved setzt sich aus Nutzziffern, Start- und Stoppzeichen zusammen. Jede Nutzziffer besteht aus fünf Elementen. (Striche und Lücken), wovon zwei Elemente breit und drei Elemente schmal sind. Die Besonderheit dieses Codes ist, dass die Ziffern an den ungeraden Stellen mit Strichen abgebildet werden. Die anderen (geraden Stellen) werden mit Lücken dargestellt, d.h. dass auch die Lücken Informationsträger sind [09].

2.2.3 Data Matrix

Die Data Matrix Codes sind rechteckige Barcodes und existieren in zwei Hauptgruppen: Code ECC 000-140 und Code ECC 200. Letzterer ist seit 1995 Standard. Die Größe des Codes ist variabel, wobei sich der Barcode aus quadratischen Symbolelementen zusammensetzt. Besonders auffällig bei diesem Code ist, dass eine Rekonstruktion des Dateninhalts noch möglich ist, wenn bis zu 25% des Codes zerstört sind [10].

2.3 RFID

2.3.1 Technologie

RFID steht für Radiofrequenz-Identifikation. Mit dieser Technik werden Daten berührungslos und ohne Sichtkontakt von einem Datenträger, dem sog. Transponder oder Tag zu einem RFID-Lesegerät und umgekehrt übertragen. Dadurch entwickelt sich ein hoher Nutzen für die Barcodeerfassung. Zum einen wird kein Sichtkontakt zu den Etiketten benötigt, zum anderen können mehrere Etiketten gleichzeitig erfasst werden.
Ein RFID System besteht aus einem RFID-Lese-/ Schreibgerät, den RFID-Tags (Transponder) und einer Software zur Auswertung der aufgenommenen Tags. Diese Tags sind in der Regel an oder in einem Produkt angebracht und enthalten einen Mircochip, um Daten zu speichern sowie eine Antenne, um die Daten zu senden. Das RFID-Lesegerät fungiert als Sende- und Empfangseinheit, indem es ein elektromagnetisches Feld erzeugt. Dieses wird von der Antenne des Tags empfangen und lädt dessen Energiespeicher auf. Dadurch wird der im Tag enthaltene Microchip aktiviert und kann über die Antenne Befehle vom Lesegerät empfangen und aussenden.

2.3.2 Transponder

Der wichtigste Teil eines RFID-Systems ist der Transponder(Tag). Er besteht aus einem kleinen Chip und einer Antenne, die in ein Gehäuse wie zum

Beispiel in einen Glasstab oder einem Klebeetikett integriert sind. Transponder sind je nach Einsatzzweck in vielen verschiedenen Größen, Bauarten und Schutzklassen erhältlich. Gängige Formen sind vor allem Klebeetiketten oder Chipkarten. Sie sind als „read only“ Version, die nur ausgelesen werden können und als „read/ write“ Ausführungen erhältlich, die sowohl ein Lesen als auch Beschreiben des Transponders ermöglichen. Häufig ist auf dem Chip eine weltweit eindeutige Nummer gespeichert, die mit Informationen in einer Datenbank verknüpft werden kann. Mittlerweile sind passive Transponder mit einer Speicherkapazität von bis zu 2000 bit erhältlich, die das Speichern zusätzlicher Informationen ermöglichen.
Das größte Unterscheidungsmerkmal bei Transpondern ist die Art der Energieversorgung. Sie sind sowohl in passiver als auch aktiver Ausführung erhältlich. Passive Transponder verfügen über keine eigene Stromversorgung und gewinnen ihre Energie durch Aufbau eines Induktionsfeldes mit Hilfe der Funksignale des Lesegeräts. Die fehlende Energiequelle bringt zwar geringere Reichweiten mit sich, ermöglicht aber kleinere Bauformen. Desweiteren sind passive Transponder wartungsfrei und zu deutlich günstigeren Preisen erhältlich. Aktive Transponder beziehen ihre Energie aus einer Batterie und können somit selbst Signale zur Datenübertragung aussenden. Aufgrund der integrierten Versorgung sind sie teurer als die passiven Gegenstücke, dafür beträgt ihre Lesereichweite bis zu 100 Meter. Sie werden vor allem zur Identifikation von Objekten benutzt, die eine hohe Lebensdauer haben und oft wiederverwendet werden [12].

3
Architekturerstellung

3.1 Einleitung

Die heutige Gesellschaft ist im hohen Maße abhängig von Software, wobei der Grad der Abhängigkeit jährlich steigt. Verstärkt übernehmen Softwaresysteme Aufgaben, welche in früheren Produktgenerationen durch Hardware oder Mechanik übernommen wurden. Neue Kommunikationstechnologien ersetzen Verkabelungen und erhöhen den Grad der Vernetzung von Systemen, während neue Displays erweiterte graphische Benutzeroberflächen ermöglichen. Der Wunsch, komplexe Anforderungen schneller und kostengünstiger bei gleichzeitig hoher Softwarequalität umzusetzen, lässt das Thema Architektur zunehmend ins Blickfeld rücken. Zukünftig wird der Softwarearchitektur eine Schlüsselstellung in Entwicklungsprozessen und Technologien zukommen und die Art, wie Software entwickelt wird, wird sich im Vergleich zu heute deutlich verändern. Während heute noch programmieren das zentrale Element des Ent-

wicklers ist, wird für den Entwickler der Zukunft die Fähigkeit,Architekturen zu erstellen, zu einem wesentlichen Berufsaspekt gehören [13].

3.2 Vorgehen zur Erstellung der Architektur

Die Erstellung einer Architektur vollzieht sich in mehreren Schritten. Es muss der Architekturentwurf vorbereitet, der Entwurf durchgeführt, die Dokumentation erstellt, die Architektur bewertet und umgesetzt werden. Bevor mit der Architekturerstellung begonnen werden kann, sind Vorbereitungen für die Erstellung zu treffen. Dazu gehört die Durchführung einer Anforderungsanalyse. Die Anforderungsanalyse liefert wichtige Eingangsinformationen für die zu entwerfende Architektur. Es handelt sich dabei um eine Anforderungsspezifikation, die sich aus funktionalen, nicht funktionalen und technischen Anforderungen zusammensetzt. Nach der Anforderungsanalyse werden die Einflussfaktoren bestimmt, die sich auf die zu erstellende Architektur auswirken oder sie beeinflussen. Durch die Bestimmung der Einflussfaktoren kann die Architektur zielgerichtet entworfen werden, sodass die Anforderungen, die an das System gestellt werden, erfüllt werden. Die ermittelten Einflussfaktoren helfen somit dem Architekten sich im Entwurfsprozess zu orientieren. Nach der Ermittlung der Einflussfaktoren erfolgt der Entwurf der Architektur durch den Architekten oder das Architektenteam. Gleichzeitig dokumentiert man den Vorgang der Architekturerstellung. Neben den Einflussfaktoren haben auch die verschiedenen Formen des Wissens, wie Basiswissen, Domänenwissen, Firmen Know-how, die Erfahrung des Architekten oder für den Entwurf verwendete Heuristiken Einfluss auf den Entwurf der Architektur. Zusätzlich werden Bewertungsmechanismen angewendet, die die Architektur zu bestimmten Zeitpunkten bestätigen. Für den Berwertungsvorgang bedient man sich verschiedener Methoden. Beispiele dazu sind die Ad-hoc Bewertung, das Early Discovery Review sowie dem umfangreichen Assessment. Die Ergebnisse des Berwertungsvorgangs fließen in den weiteren Architekturentwurf. Die Bewertungsmaßnahmen nehmen somit eine Kontrollfunktion wahr, sodass die Architektur Schritt für Schritt den bestehenden Anforderungen angepasst wird. Der Prozess des Entwerfens

der Architektur und des Bewertens stellt somit einen iterativen Schritt in der Entwurfsphase dar. Steht der Architektur-Entwurf, d.h. die Architektur wurde dokumentiert und bewertet, kann mit der Umsetzung der Architektur begonnen werden. Die Dokumentation und Bewertung der Software-Architektur dient dabei als Grundlage für die Implementierung. Der Entwurf der Architektur endet aber noch nicht mit der Implementierung, sondern muss weiterhin überarbeitet und verbessert werden. Somit ist der Architekturentwurf ein iterativer und inkrementeller Entwicklungs-Prozess, bei dem die einzelnen Schritte mehrmals durchlaufen werden müssen, um die Architektur schrittweise aufzubauen und an die festgesetzten Anforderungen anzupassen [12].

3.3 Abgrenzung zu existierenden Systemen

Dieses Kapitel beschreibt einige bestehende Systeme zur Barcodeerfassung. Aus den dadurch gewonnenen Erfahrungen werden Anforderungen für das zu entwickelnde System abgeleitet, die in den folgenden Kapiteln den Entwurf der Architektur bestimmen.
Die meisten in der Wirtschaft verkauften MDE-Geräte werden ohne Barcode-Software ausgeliefert. Stattdessen beinhalten die mitgelieferten Softwarepakete eine Vielzahl von Bibliotheken und Dokumentationen, welche exakt die Schnittstellen der Hardware beschreiben. Sie beinhalten nicht nur Details, um den Scanner anzusprechen, sondern auch Beispielcode für den Zugriff auf verschiedene Schnittstellen.
NordicID stellt für das Gerät *NordicID morphic* eine 234-seitige Dokumentation [14] zur Verfügung. Damit lassen sich alle Hardwareattribute ansprechen und ändern. Die Auswahl beschränkt sich dabei nicht nur auf den Scanner, sondern auch auf Display, Tastatur, Beleuchtungen, WLAN und RFID. Für Entwickler kann dies sehr interessant sein, da die Funktionalität und Individualität eines Programms mit sehr einfachen Mitteln angepasst werden. Allerdings ist darauf zu achten, dass diese angepasste Software nur auf einem NordicID-Gerät verfügbar ist und daher nicht für verschiedene Gerätetypen zur Verfügung gestellt werden kann [15].

Andere Hersteller liefern die MDE-Geräte mit einem minimalen Umfang an Software aus. Dies beläuft sich in der Regel nur auf Barcodeerfassung und den Export der aufgenommenen Daten. Der Vorteil eines solchen Systems ist der geringe Preis. Da aber die Speicherkapazität dieser Geräte stark begrenzt ist, bieten sie sich nur für kleinere Unternehmen an, die ein- bis zweimal im Jahr eine Inventur durchführen [16].
Eine dritte Variante liefert die IDS GmbH mit dem Gerät *Handyscan 8000 C*. Dabei handelt es sich um eine Kombination der bestehenden Varianten. Zusätzlich zu einer vorhandenen Software wird dem Barcodescanner ein Programmgenerator mitgeliefert. Mit diesem Generator lassen sich auch ohne Programmierkenntnisse einfache Programmmodule entwickeln [17].
Im Rahmen einer Praktikumsarbeit wurde bereits ein etwas dynamischeres Konzept entwickelt und implementiert. Die Grundidee war eine Software für verschiedene Geräte verfügbar zu machen, bzw. die Anpassung an die Gerätetreiber auf ein Minimum zu reduzieren. Während der Analyse der verschiedenen Gerätetypen wurde festgestellt, dass es eine Vielzahl verschiedener Displaygrößen gibt. Von grafischen 2“ bis 5“ Displays mit verschiedenen Auflösungen, über LCD -Displays, bis zu Geräten ohne Displays ist alles vorhanden. Aus diesem Grund wurde gefordert, ein Paket zu entwickeln, welches nur grafische Komponenten enthält, um diese gegebenfalls auszutauschen. Dasselbe Prinzip wurde bei der Datenhaltungsschicht angewandt. Diese wurde getrennt vom Programmcode entwickelt, um ohne Zeitaufwand in eine andere Technologie getauscht zu werden.
Aus diesen Anforderungen wurde eine 4-Schicht-Architektur entwickelt.

Während der Entwicklung auf drei verschiedenen Geräten hat sich diese Architektur als geeignet bewährt. Neue Scanner ließen sich ohne Probleme einbinden, allerdings nahm die Einarbeitung in die Herstellerschnittstellen einige Zeit in Anspruch. Die grafischen Oberflächen haben sich sogar noch besser bewährt als erwartet. Diese mussten trotz unterschiedlicher Displaygrößen nicht geändert werden, da sie in der Regel dasselbe Seiten- und Höhenverhältnis haben.

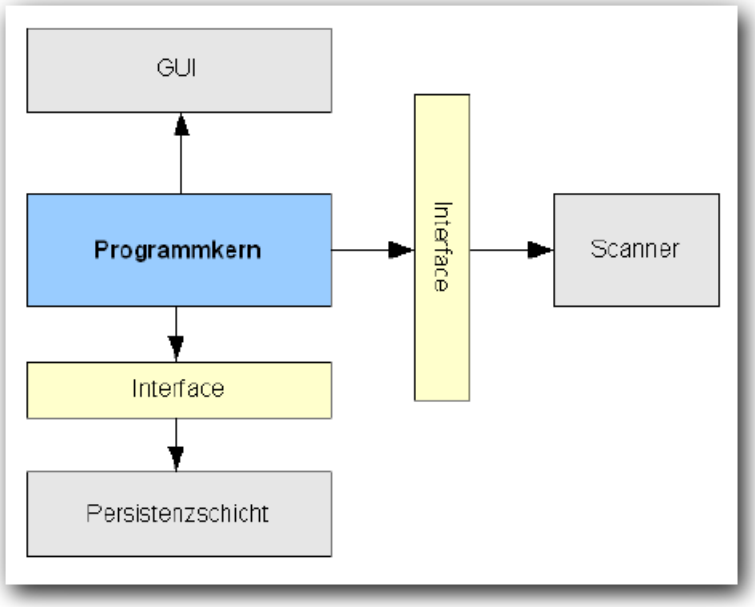

Abbildung 3.1: Vier-Schicht-Architektur

3.4 Anforderungsanalyse

3.4.1 Anforderungen

Die Erfahrungen des zuvor entwickelten Systems bilden die Grundlage für die neue erweiterbare Architektur. Es soll weiterhin einen seperaten Programmkern geben. Allerdings wird dieser nicht mehr den Code der Barcodemodule enthalten, sondern grundlegende Funktionalitäten. Das können Algorithmen zur Barcodeerkennung, Prüfsummenberechnung oder Programmsteuerung sein. Die größte Änderung der Architektur liegt in der Einführung einer PlugIn-Struktur, um ein dynamisches Hinzufügen von Modulen zu ermöglichen. Dies soll gewährleisten, dass ein Modul unabhängig von dem restlichen Code entworfen werden kann. Die Erkennung, ob es sich um ein gültiges Modul handelt, muss von dem Programmkern verwaltet werden, da dieser auch die Aufgabe hat, die Module auf einer zentralen GUI darzustellen. Desweiteren soll der Kern eine Anbindung zu einer Datenhaltungsschicht erhalten. Im Unterschied

zu der 4-Schichtarchitektur soll aber die Möglichkeit in Betracht gezogen werden, verschiedene Datenhaltungstechnologien anzusprechen.
Der Scanner wird weiterhin von dem Modul angesprochen, von dem es auch genutzt wird. Da sich dieses jetzt innerhalb einer PlugIn-Struktur befindet, muss auch von diesem Punkt die Scanneranbindung erfolgen.
Eine weitere Neuerung wird die Abkapselung der Kommunikation mit einem externen Gerät. Während der vorangegangen Entwicklung hat sich gezeigt, dass sich die Softwarehandhabung für den Nutzer sehr vereinfacht, wenn die Daten mit nur einem Knopfdruck auf den PC übertragen werden können. Daher soll die Möglichkeit eingeräumt werden, Daten zu exportieren, zu importieren oder eine direkte Verbindung zu einem Warenwirtschaftssystem aufzubauen.

3.4.2 Fachliches Modell

Basierend auf den Erfahrungen des zuvor entwickelten Systems, lassen sich die meisten Merkmale und Anforderungen übernehmen. Innerhalb des fachlichen Modells, oder auch fachliche Architektur genannt, werden nur die wesentlichen Klassen und deren Beziehungen zueinander dargestellt.
Eine wesentliche Änderung zu dem bestehenden System ist die Trennung des funktionalen Codes aus dem Programmkern und die Integration der Modul-GUI in eine PlugIn-Struktur. Die Klassen haben dabei folgende Bedeutung:

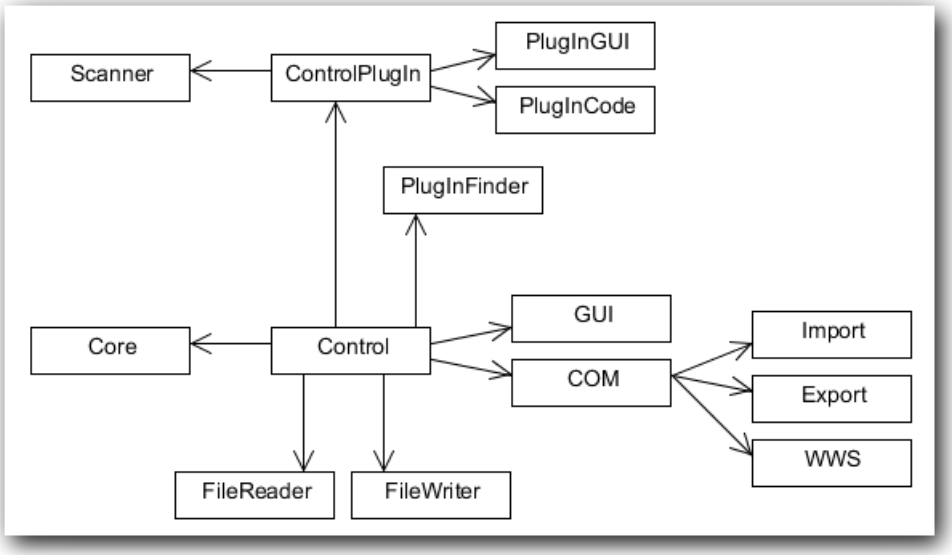

Abbildung 3.2: Fachliches Modell

- Control: Steuerlogik zur Koordination der verbundenen Klassen
- Core: Funktionen zur Barcodeerfassung, die von jedem Modul genutzt werden können (auch Ablagemöglichkeit für häufig genutzte graphische Oberflächen)
- GUI: Graphische Oberfläche für das Hauptprogramm und Darstellung der PlugIn-Module

- PlugInFinder: Sucht nach PlugIn-Modulen zur Barcodeerfassung
- ControlPlugIn: Steuerlogik innerhalb eines PlugIn-Moduls
- PlugInGUI: Graphische Elemente eines Moduls
- PlugInCode: Funktionalität eines Moduls
- Scanner: Gerätetreiber zum Zugriff auf den Scanner
- COM : Kommunikationsschnittstelle zu einem externen Gerät (PC, Warenwirtschaftssystem, anderes MDE-Gerät
- Import: Importiert Daten von einem externen Gerät
- Export: Exportiert Daten von einem externen Gerät
- WWS : Zugriff auf ein Warenwirtschaftssystem
- FileReader: Lesen der vorhandenen Barcodedaten
- FileWriter: Schreiben der gescannten Barcodedaten

3.5 Einflussfaktoren

Ziel der Analyse der Einflussfaktoren ist es, einzelne Aspekte des Systems zu bestimmen und gegeneinander abzuwägen. Dabei werden technische, organisatorische und Produktfaktoren berücksichtigt.

3.5.1 Technologische Faktoren

Durch die Auswertung der technologischen Faktoren lassen sich wichtige Entscheidungen im Bereich der Hardware- und Softwaretechnologien spezifizieren. Gerade diese technologischen Faktoren sind sehr wichtig, da nahezu jeder Hersteller von MDE-Geräten andere Konzepte und Vorgehensweisen verwendet.
Ein grundlegender Punkt ist das vorinstallierte Betriebssystem eines MDE-Gerätes. Zur Zeit existieren viele Geräte mit einem Windowsbetriebssystem

wie z.B. WinMobile03, WinCe 5.0 oder WinCe 6.0 . Bei diesen Geräten kann eine .NET oder eine Java-SDK als Entwicklungsumgebung herangezogen werden(siehe auch Organisatorische Faktoren). Dabei sind allerdings einige Punkte zu beachten: Die Implementierung auf MDE-Geräten erfolgt nicht mit den üblichen Frameworks für PC´s, sondern mittels einer vereinfachten Version der entsprechenden Sprache. Microsoft stellt für sein .Net-Framework eine Compact-Edition von ca. 9 MB (aktuelle Version 3.5), während SUN eine Micro Edition von ca. 7MB entwickelte. Für Javaanwendungen gibt es zusätzlich einige Emulatoren, die in der Lage sind, Java-PC-Code (mittels Java-SDK erstellt) zu simulieren. Von dieser Vorgehensweise wird jedoch dringend abgeraten, da die Performanceverluste so erheblich sind, dass ein wirtschaftliches Arbeiten nicht möglich ist.
Ein weiteres geeignetes Betriebssystem für mobile Geräte ist das von Google entwickelte System Android. Dieses System basiert auf einem Linux-Kernel und lässt sich damit gut für eine Java-Architektur nutzen.
Unabhängig von der Wahl einer Programmiersprache ist es wichtig, eine Entwicklungsumgebung zu wählen, mit der sich einzelne Schichten separat entwickeln lassen können. Eine Möglichkeit ist jede Schicht der Architektur als einzelnes Projekt zu implementieren. Diese Vorgehensweise hat den Vorteil, dass einzelne Module und Schichten getauscht, erweitert oder neu entwickelt werden können, ohne auf den verbleibenden Code zuzugreifen. Dafür müssen allerdings klare Schnittstellen definiert und dokumentiert werden.
Ein weiterer technologischer Faktor, der berücksichtigt werden muss, ist die Hardware eines Gerätes. Der Zugriff auf diese Komponenten wird vom Hersteller durch Schnittstellen festgelegt. Wenn die Software einem anderen Gerät zur Verfügung gestellt werden soll, ist daher eine neue Entwicklung des Hardwarezugriffs nötig. Aus diesem Grund muss auch für den Hardwarezugriff eine eigene Architekturschicht in Betracht gezogen werden.

3.5.2 Produktfaktoren

Produktfaktoren haben einen großen Einfluss auf die Softwarearchitektur. Dabei werden funktionale und nichtfunktionale Anforderungen berücksichtigt. Zu der ersten Kategorie gehören Merkmale wie Portierbarkbeit, Testbarkeit und Wiederverwendbarkeit [12]. Die Abkapselung des Scanners von dem Hauptprogrammkern beinhaltet den ersten Schritt einer hohen Portierbarkeit des Programms auf andere MDE-Geräte. Der nächste Schritt besteht in der Trennung der Persistenzschicht. Damit wird gewährleistet, dass für jedes System eine geeignete Datenhaltung in Betracht gezogen werden kann. Für windowsbasierende Systeme kann ein geeignetes MS -Datenbanksystem verwendet werden, während sich für linuxbasierende Systeme ein Oracle-DBMS anbietet. Desweiteren kann diese Schicht zur Verwaltung mehrerer Speichertechnologien (DBMS, XML, CSV) genutzt werden, insofern dies von einer geeigneten Schnittstelle unterstützt wird.
Ein weiterer Punkt der Produktfaktoren und Grundsatz der OOP ist die Wiederverwendbarkeit. Dabei wird angestrebt, möglichst viel bestehenden Code für andere oder ähnliche Projekte erneut zu nutzen. Gerade im Bereich der Barcodeerfassung ist es nicht einfach, dieses Konzept konsequent umzusetzen, da jedes Unternehmen individuelle Wünsche über die Funktionsweise der Software fordert. Eine Trennung von individuellem Programmcode und Basiscode ist daher notwendig. Die Schicht mit dem Programmcode enthält die graphischen Oberflächen und Funktionen für den Anwender. Hier muss besonders auf die Kapselung geachtet werden, um nicht alle Funktionen in eine Bibliothek oder ein Paket zu implementieren. Es bietet sich an, für jedes Modul (Inventur, Preisänderung, etc.) ein einzelnes Projekt anzulegen, um dieses dynamisch zur Laufzeit zu laden und bereitzustellen. Aber auch innerhalb eines Modulprojektes ist auf eine gewisse Kapselung zu achten: Kleine Projekte sollten nach dem gängigen MVC-Prinzip aufgebaut werden, während größere Projekte eine eigene Sub-Architektur erhalten.
Der Basiscode sollte in einem Programmkern eingekapselt werden, da er mit dieser Vorgehensweise von jedem Modul genutzt werden kann. Der Kern muss

dabei in seine einzelne Aufgabenbereiche getrennt werden, wie zum Beispiel Algorithmen zur Barcodeerfassung, die Bereitstellung vordefinierter GUI´s und die Zugriffsverwaltung zu einer Datenhaltungsschicht.

3.5.3 Organisatorische Faktoren

Die organisatorischen Faktoren spezifizieren das Management, Mitarbeiter, Zeitpläne und Budgets eines Projektes. Aus diesem Grund hat der Softwarearchitekt nur begrenzte Möglichkeiten, diese Faktoren kurzfristig zu ändern. Ein grundlegender Faktor ist das Know-how der Mitarbeiter. Diese müssen sich gut mit den geforderten Technologien und Programmiersprachen auskennen. Ist das nicht der Fall, bleiben nur die Möglichkeiten auf diese Technologien zu verzichten (Funktionalitätsverlust!), die Mitarbeiter weiterzubilden (Zeitfaktor!) oder externe Mitarbeiter einzustellen (Kostenfaktor !). Die Wahl der Entscheidung hängt daher von dem Entwicklungsbudget und dem Zeitplan ab. Diese müssen vor allem die Kosten für benötigte MDE-Geräte, eventuelle Weiterbildungsmaßnahmen und den Liefertermin für den Kunden enthalten.

3.5.4 Zusammenfassung

Um die oben genannten Einflussfaktoren noch genauer zu spezifizieren, werden folgende Faktoren für das weitere Projekt festgelegt:

- Betriebssystem: WinMobile / WinCe
- Programmiersprache: C#
- Entwicklungsumgebung: MSVS2005
- Kenntnisse Barcodeerfassung: gut
- Kenntnisse Sofwareentwicklung: gut

Technologische Faktoren	**Flexibilität und Veränderbarkeit**	**Einfluss**
T1. Betriebssystem		
Als Betriebssystem wird WinMobile oder WinCe eingesetzt.	Das Betriebssystem kann sich in zukünftigen Versionen ändern.	Designentscheidung
T2. Paketstruktur		
Das System wird in einzelne Pakete gekapselt, um kritische Teile schnell auszutauschen.	Jedes Paket wird getrennt von den anderen entwickelt und über Schnittstellen eingebunden. Nachträgliche Änderungen/Erweiterungen lassen sich ohne Probleme durchführen.	Designentscheidung
T3. Hardware		
Jedes Gerät besitzt einen anderen Hardwarezugriff.	Die Hardwaretreiber werden über definierte Schnittstellen eingebunden und können für jedes Gerät getauscht werden.	Designentscheidung
T4. Softwaretechnologien		
P4.1. Entwicklungsumgebung		
Es wird das .Net Framework verwendet.	Die .Net-Umgebung stellt bereits eine Vielzahl von Klassen bereit, sodass eine schnelle Entwicklung der Software garantiert werden kann.	Designentscheidung
P4.2. Programmiersprache		
Es soll die .Net-Entwicklungsprache C# verwendet werden.	Die Wahl der Sprache kann nicht verändert werden.	Designentscheidung

Tabelle 3.1: Technologische Faktoren

Produkt-faktoren	**Flexibilität und Veränderbarkeit**	**Einfluss**
P1. Funktionale Anforderungen		
P1.1 Datenhaltung		
Das Paket stellt über eine Schnittstelle den Zugriff auf verschiedene Ablagemöglichkeiten.	Der Zugriff auf die Datenhaltung kann zur Laufzeit geändert werden.	Designentscheidung
P1.2. Module		
PlugIn-Module werden als DLL entworfen.	Jedes Modul kann unabhängig von dem restlichen Code entworfen, getestet und verändert werden.	Designentscheidung
P1.3. Kommunikation		
Es wird eine Anbindung zu externen Geräten entworfen.	Auf geänderte Exportbedingungen kann schnell reagiert werden.	Designentscheidung
P1.4. Barcodefunktionen		
Alle Barcodefunktionen werden zentral verwaltet.	Funktionen können beliebig erweitert und verändert werden. Durch die zentrale Verwaltung stehen sie allen Modulen zur Verfügung.	Designentscheidung

Tabelle 3.2: Produktfaktoren

Organisatorische Faktoren	**Flexibilität und Veränderbarkeit**	**Einfluss**
O1. Zeitplan		
Der Liefertermin steht fest.	Es gibt kein Verhandlungsspielraum.	Designentscheidung, Zeitplan
O2. Budget		
O2.1. Hardware		
MDE-Geräte werden für die Entwicklungsdauer von dem Kunden bereitgestellt.	Es fallen keine Hardwarekosten an.	Budgetentscheidung
O2.2. Software		
Es wird eine .Net-Entwicklungsumgebung benötigt.	Die benötigte Software ist vorhanden oder frei verfügbar.	Budgetentscheidung
O3. Kenntnisse		
O3.1. Barcodes		
Es werden Kenntnisse über die Funktionsweise von Barcodes benötigt.	Kenntnisse sind im erforderlichen Umfang verfügbar.	Budgetentscheidung, Zeitplan
O3.2. Softwareentwicklung		
Es werden Kenntnisse über MSVS, .Net CF und ggf. Datenbanken benötigt.	Kenntnisse sind im erforderlichen Umfang verfügbar.	Budgetentscheidung, Zeitplan

Tabelle 3.3: Organisatorische Faktoren

3.6 Systemkontext

Aus den genannten Einflussfaktoren lässt sich ein erster Grobentwurf (Systemkontext) der Architektur ableiten. Dabei werden funktionale, nichtfunktionale und technische Anforderungen berücksichtigt und ein geeigneter Architekturstil gewählt.

- Hardware: Im Laufe der Entwicklung hat sich gezeigt, dass eine Schicht für die Scannerverwaltung nicht ausreichend ist, da in der Regel mehrere Merkmale der Hardware angesprochen werden müssen. Daher wird diese Schicht alle Funktionen zur Ansteuerung der Hardware übernehmen. Neben dem Scannerzugriff werden Zugriffsmethoden für das Display(Vollbildmodus, Farbtiefe, Auflösung, etc), Tastatur(Tastendruckevents, Tastenkonstanten, etc), Datenübertragung(LAN, WLAN, Bluetooth, etc), Utilitys(Seriennummer, CPU-Info, Geräteadressen, etc) oder RFID-Funktionen zur Verfügung gestellt.

- Plugin: Die Pluginschicht beinhaltet die Module zur Barcodeerfassung. Jedes Modul muss dabei ein vorgefertigtes Interface implementieren, um als Plugin erkannt zu werden. Um diese Struktur umzusetzen, wird jedes Modul als einzelnes Projekt entwickelt. Die Erfahrungen aus dem vorherigen Projekt sowie die Displayeigenschaften der MDE-Geräte haben gezeigt, dass es nicht notwendig ist, eine eigene Schicht für die grafische Oberfläche zu entwickeln. Aus diesem Grund wird jedes Modul seine eigene GUI beinhalten.

- Kern: Der Kern wird alle nötigen Bibliotheken zu Barcodeerfassung enthalten. Da diese Funktionen später von jedem Modul genutzt werden können, besteht die Möglichkeit, häufig genutzte GUI´s in dieser Schicht abzulegen. Desweiteren wird in dieser Schicht die Anbindung der Plugins verwaltet.

- Datenhaltung: Die Datenhaltungsschicht verwaltet den Zugriff auf alle zu speichernden oder zu ladenden Daten. Der Kern wird dabei ein entsprechendes Interface ansprechen, das den Zugriff auf Ablagemöglichkeiten (DB, XML, CSV, etc) gewährt.

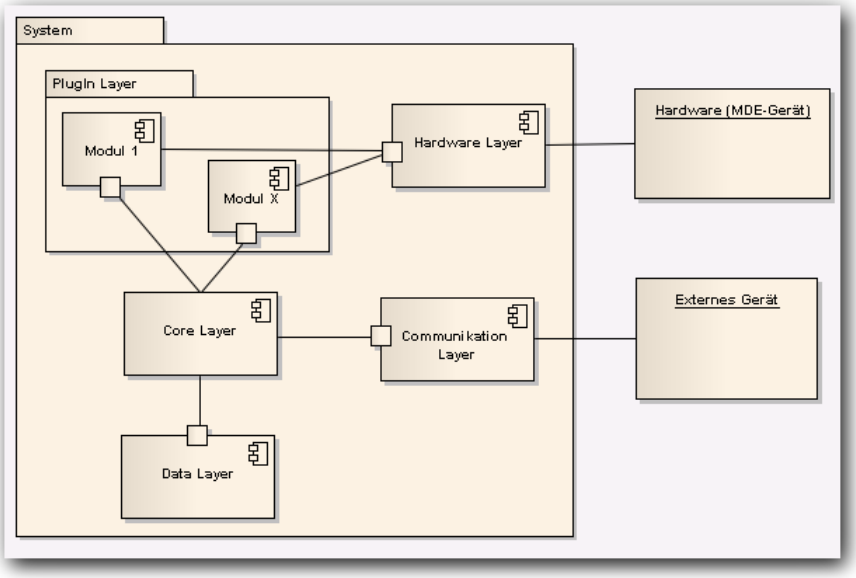

Abbildung 3.3: Systemkontext

Resultat:
Aus den Anforderungen lässt sich ein klarer Architekturstil erkennen: Der *Layer* [18]. Ziel des Stils ist es, eine niedrige Abstraktionsebene in eine höhere Abstraktionsebene zu heben und die dabei entstehende Funktionalität an andere Anwendungen weiter zu geben. Zusätzlich kann der Architekturstil Layer die Klassenbibliothek strukturieren und in verschiedene Arbeitsbereiche unterteilen.

3.7 Schnittstellen

Softwarearchitekturen werden aus Schichten, Subsystemen und Komponenten zusammengesetzt. Diese Elemente können durch verschiedene Entwickler getrennt implementiert werden, um damit einen schnelleren Projektabschluss zu erreichen. Solches Vorgehen hat aber einen grundlegenden Nachteil: Schon zu Beginn der Entwicklung müssen die Schnittstellen zwischen den Schichten definiert werden. Werden diese Schnittstellen nur unzureichend umgesetzt, müssen sie im Laufe der Entwicklung neu angepasst werden. Das hat zur Folge, dass auch die angrenzenden Elemente angepasst werden müssen und der Projektabschluss sich verzögert [12].

> *„Eine Schnittstelle ist ein Vertrag zwischen einer Komponente, die eine bestimmte Funktionalität benötigt, und einer Komponente, die diese Funktionalität bereitstellt.“*
>
> J. Bosch [19]

- Hardware: Die Hardwareschnittstelle stellt die anspruchsvollste und komplexeste Schnittstelle der Architektur dar. Sie muss jeden Hardwarezugriff entgegennehmen und bearbeiten. Eine Möglichkeit für diese Umsetzung ist die Analyse aller Geräte und Einarbeitung in die Gerätetreiber. Aus den gewonnenen Kenntnissen kann eine Struktur entwickelt werden, mit der eine Art Universaltreiber entworfen werden kann. Allerdings wird es nicht möglich sein, 100% der Funktionalität zu übernehmen, ohne eine Schnittstelle mit mehreren hundert Methoden zu entwerfen. Eine alternative Idee ist die Umsetzung von flexiblen Lese- und Schreibmethoden. Dabei wird nur ein Hardwaretyp, zB. Scanner, Display,etc. und eine Aufgabe an eine Funktion übergeben. Die Schnittstelle wird dann entscheiden, ob die Aufgabe ausführbar ist oder ob ein Fehlercode zurückgeliefert werden muss.

```
1 //Funktion zum Setzen und Initialiesieren von Geräteeigenschaften
2 public int deviceSetter(int deviceType, String task);
3
4 //Funktion zum Abfragen von Geräteinformationen
5 public Object deviceGetter(int deviceType, String task);
```

- Datenhaltung: Die Schnittstelle der Datenhaltung hat die Aufgabe, alle anfallenden Informationen in ein Datenformat zu speichern und zu laden. Da in der Regel dieses Format zu Beginn der Entwicklung noch nicht bekannt ist, oder noch geändert wird, muss auch hier ein dynamisches Konzept angewandt werden. Dieses kann ähnlich aufgebaut werden wie die Hardwareschnittstelle: Als erstes Argument wird ein Datenformat(DB, XML, etc) übergeben, um die entsprechende Ablagemöglichkeit einzubinden. Das zweite Argument enthält zur Spezialisierung einen Datei- oder Tabellennamen und das dritte Argument eine Datenmatrix mit den gesammelten Informationen.

```
1 //Funktion zum erstellen einer Datenquelle
2 public int initData(int dataType, String name, String[] colName);
3
4 // Schreiben von einer Datenzeile
5 public int writeDataLine(int dataType, String name, String[] data);
6
7 // Löschen einer Datenzeile
8 public int writeDataLine(int dataType, String name, String[] data);
9
10 //Lesen einer kompletten Tabelle
11 public Object[][] readData(int dataType, String[] data);
```

- Modulschicht: Jedes Modul muss einem bestimmten Format entsprechen, um auch als gültiges PlugIn-Modul erkannt zu werden. Da dieses Format sowohl dem jeweiligen Modul als auch dem Modul-Finder im Kern bekannt sein muss, wird ein seperater Datentyp(Klasse) entworfen. Dieser Typ enthält einen Namen für das Modul sowie eine Funktion, um das Modul anzuzeigen. Eine Registrierfunktion übernimmt diesen Typ und aktiviert die gültigen PlugIns.

```
1 //Registrieungsfunktion für PlugIns
2 bool Register(IPlugin ipi);
3
4 public interface IPlugin
5 {
6         string Name{get;set;}
7         void Show();
8 }
```

- Kommunikationsschicht: Diese Schnittstelle verwaltet die Kommunikation zwischen einem MDE-Gerät und einem anderen Endgerät (PC, anderes MDE-Gerät, etc). Dabei ist es ausreichend, eine Funktion für den Import und eine Funktion für den Export bereitzustellen. Um eine hohe Flexibilität zu gewährleisten, empfiehlt es sich, neben der Datenmatrix noch eine Argumentliste zu übergeben. Dadurch können jederzeit beliebig viele Parameter genutzt werden.

```
1 //Importfunktion
2 public Object[][] import(String[] args);
3
4 // Exportfunktion
5 public Boolean export(String[] args, Object[][] dataMatrix);
```

3.8 Entwurf

Nachdem alle Anforderungen analysiert, Einflussfaktoren bestimmt und Schnittstellen definiert wurden, lässt sich aus dem fachlichen Modell und dem Systemkontext ein Architekturmodell erstellen. Der Aufbau erfolgt dabei mit dem bereits analysierten Layer-Stil, um die einzelenen Funktionsabschnitte zu trennen. Die Verbindung zwischen den Schichten wird dabei über Schnittstellen realiesiert. Dabei wird gekennzeichnet welche Schicht diese Schnittstelle enthält und welche diese nutzt.
Jede Schicht wird eine gewisse Anzahl an Klassen enthalten, die für die Entwicklung des Prototypen notwendig sind. Für die Datenhaltung stehen verschiedene Muster wie z.B. Datenbanken oder XML-Strukturen zur Verfügung, allerdings werden nur die benötigen Muster eingebunden.

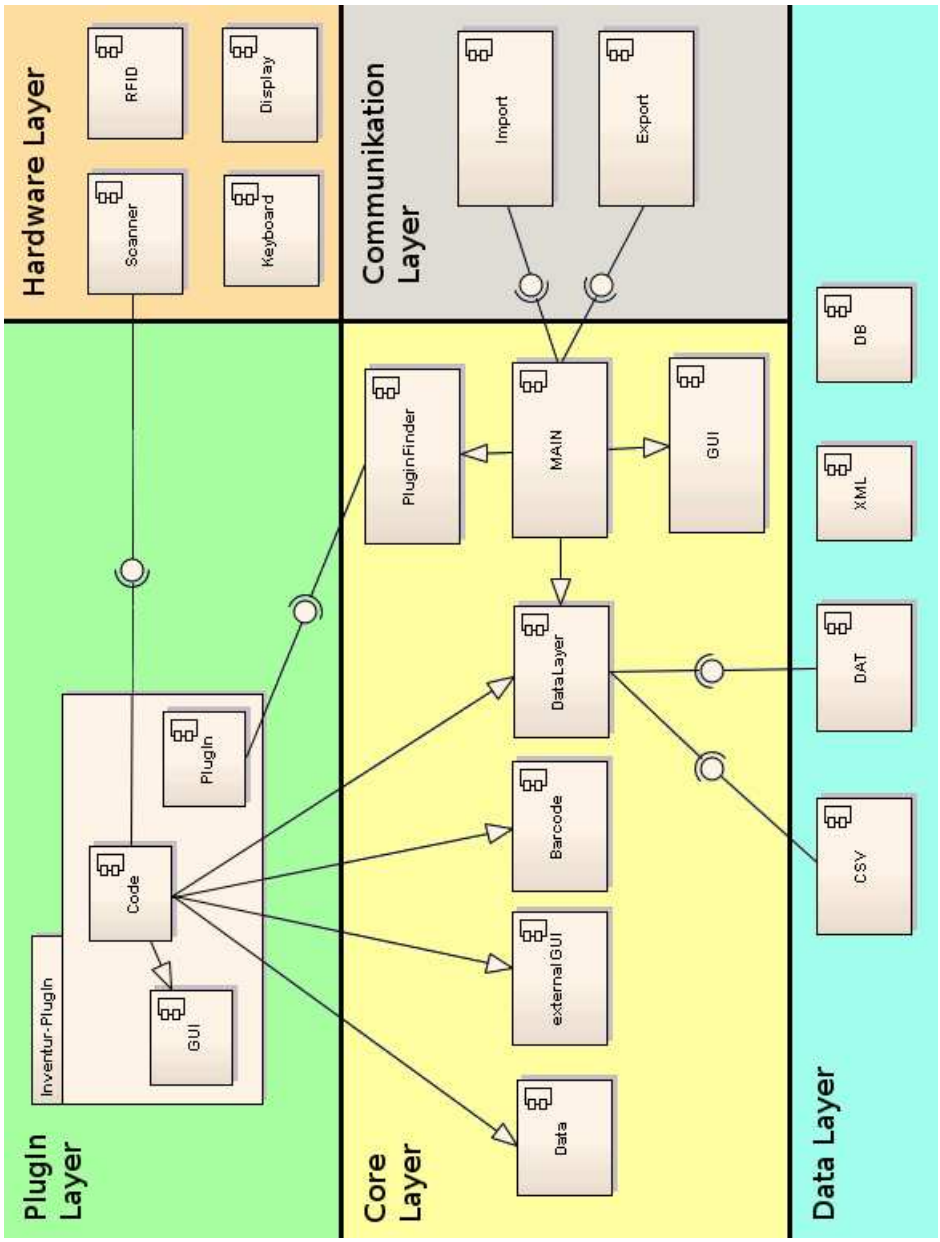

Abbildung 3.4: Plugin-Architektur

3.9 Dokumentation

„A documentation package for a software architecture is composed of one or more view documents and documentation that explains how the view relate to another, introduce the package to its readers, and guides them through it.“
Paul Clement [20]

Die Dokumentation der Architektur besteht aus einer Vielzahl von verschiedenen Dokumenten. Eine Voraussetzung für die Dokumentation der Architektur besteht in der Festlegung der Struktur der Architekturbeschreibung. Das Wesentliche an der Erstellung der Dokumentation ist, dass die Dokumente einem konkreten Zielpublikum, wie zum Bsp. die Benutzer und die Entwickler, sowie einem konkreten Zweck, wie Handlungs- bzw. Implementierungsanweisung usw., gewidmet sein sollen. Die Themen der Dokumente betreffen die Liste der Einflussfaktoren, die Liste der Risiken und potenziellen Lösungsstrategien, Hinweise zum Lesen der Dokumentation, Architektursichten und die Bewertung des aktuellen Entwurfs. Die Liste der Einflussfaktoren enthält alle systematisch erfassten Einflüsse, die auf das System einwirken und gleichzeitig dient sie als Vorlage für die Bewertung und Priorisierung der Anforderungen. Die Liste der Risiken und potenziellen Lösungsstrategien haben Bedeutung für den Architekten und den Projektleiter. Dem Architekten dient sie zur Auswahl von Konzepten für einen ersten Systementwurf, dem Projektleiter dient die Liste als Hilfestellung für die Bewertung der Projektrisiken. Die Architektursichtendokumentation dient dem Architekten sowie dem Entwicklerteam als Kommunikationsmittel. Die Beschreibung der Architektursichten, wie auch der Einflussfaktoren, Risiken und Lösungsstrategien, stellt das Kernwissen des Projektes dar, welches zudem für Folgeprojekte gewinnbringend eingesetzt werden kann. Um die bisherige Arbeit einzuschätzen wird der aktuelle Entwurf bewertet. Die Bewertung des aktuellen Entwurfs dient somit für alle beteiligten Personen als objektive Beurteilung [20], [21].

4
Bewertung und Implementierung

4.1 Umfangreiches Assessment

Das umfangreiche Assessment hat die Aufgabe, die Richtigkeit und Angemessenheit der Architektur nochmals zu überprüfen, bevor mit der Implementierung begonnen werden kann. Für die Durchführung der Bewertung wendet man die sogenannte Architectur Tradeoff Analysis Methode (ATAM) an. Es handelt sich dabei um eine szenariobasierte Vorgehensweise. Die ATAM-Methode bzw. der Kern der ATAM-Methodik lässt sich in 6 Schritte unterteilen, welche nachfolgend durchlaufen werden [12].

Schritt 1-3: Präsentation

Die ersten drei Schritte befassen sich mit der Präsentation des Projektes. Die beteiligten Personen in diesen Phasen sind der Teamleiter des Bewertungsteams, der die ATAM-Methode vorstellt, der Projektleiter, der das Geschäftsziel des Projektes vorstellt und der Architekt, welcher eine Kontextsicht des Systems sowie die das Zusammenspiel der internen Komponenten vorstellt.

Schritt 4: Architekturansätze identifizieren

Der Architekturstil Layer resultiert aus der Verwendung der Klassenbibliothek heraus. Ziel ist es, eine niedrige Abstraktionsebene in eine höhere Abstraktionsebene zu heben und die dabei entstehende Funktionalität an andere Anwendungen, die sich mit der Klassenbibliothek verbinden, weiter zu geben. Zusätzlich soll der Architekturstil Layer die Klassenbibliothek strukturieren und in verschiedene Arbeitsbereiche unterteilen.

Schritt 5: Utility Tree aufbauen

Nach dem die Architekturansätze ausgewertet wurden, werden anhand der ermittelten Einflussfaktoren ein Utility Tree angefertig. Da die Architektur im Wesentlichen durch die Einflussfaktoren bestimmt worden ist, wird der Utility Tree anhand dieser Anforderungen erstellt. Dabei werden die Faktoren in verschiedene Ebenen geschachtelt, wobei ein Faktor ein Blatt des Baumes darstellt. Diese werden mittels zweier Dimensionen priorisiert (niedrig, mittel, hoch) : Die erste gibt an wie wichtig dieser Faktor für den Erfolg des Systems ist. Die zweite wie schwierig es ist, das Szenario in dem System umzusetzen. Diese Priorisierung wird vom Architekten des Systems vorgenommen und geben an worauf sich das Bewertungsteam im weiteren Verlauf konzentrieren muss. Alle Anforderungen mit den Bewertungen (hoch, hoch), (hoch, mittel) oder (mittel, hoch) müssen analysiert und die Risiken abgewägt werden, während Faktoren mit einer geringen Bewertung aus Kostengründen entfallen.

Das Resultat bildet der nachfolgende Utility Tree.

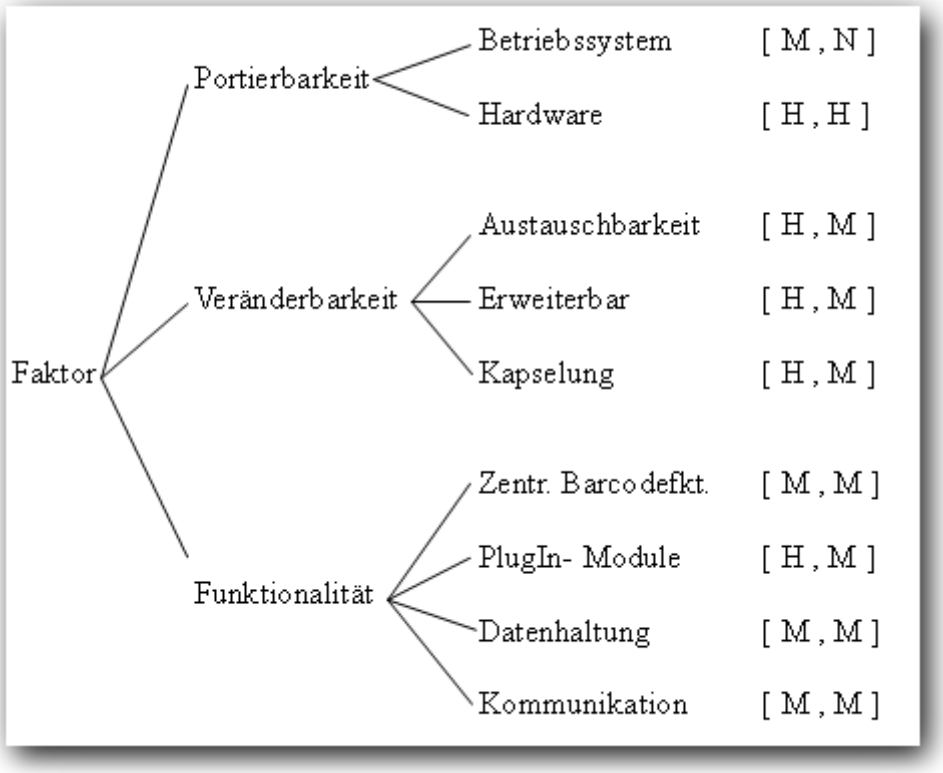

Abbildung 4.1: Utility-Tree

Schritt 6: Szenario-Brainstorming:

Werden im Szenario Brainstorming weitere Anwendungsszenarien gefunden, welche im bisherigen Konzept nicht berücksichtigt worden sind, erfolgt hier eine erneute Bewertung der Architektur. So kann es sein, dass die neu gefundenen Anwendungsszenarien mit dem bisherigen Architekturstil nicht harmonieren und dass dadurch die geforderten Aufgaben nicht erfüllt werden können. Treten solche Aspekte auf, müssen Optimierungen der vorliegenden Architektur vorgenommen werden, bevor mit der Umsetzung der Architektur begonnen werden kann.
Da keine weiteren Anwendungsszenarien gefunden wurden, entfällt eine weitere Analyse der Architekturansätze und es kann mit der Umsetzung der Architektur begonnen werden.

4.2 Prototyp

Im folgenden Kapitel wird die Vorgehensweise zur Umsetzung der Architektur erläutert. Dafür wird ein Prototyp entwickelt, der die grundlegenden Funktionen beinhaltet.
Das erste Projekt, welches entwickelt wird, ist eine Bibliothek mit allen geforderten Schnittstellen. Durch dieses Vorgehen werden Probleme in der laufenden Entwicklung vermieden, da gleich zu Beginn auf die bestehenden Definitionen zurückgegriffen werden kann. Wurden alle Schnittstellen in eine Bibliothek verpackt, kann mit dem Programmkern begonnen werden.
Der Kern ist das einzige Projekt, dass als ausführbare Datei(.exe) compiliert wird. Alle anderen Programmteile werden als Klassenbibliothek (.dll) entwickelt und eingebunden. Zuerst wird eine einfache graphische Oberfläche erstellt, auf der die Module abgebildet werden können. Diese werden von einem PlugIn-Finder erkannt, der alle DLL's in einem Verzeichnis analysiert. Dabei wird überprüft, ob eine DLL eine Klasse mit dem Namen „PlugIn“ enthält. Um eine korrekte Anbindung der Klasse zu gewährleisten wird ein entsprechendes

Interface(siehe Punkt Schnittstellen) eingebunden und implementiert. Gefundene Module werden mit ihrem Namen als Button auf der GUI angezeigt. Für den kommerziellen Gebrauch sollte hier eine andere Darstellungsweise gewählt werden, da eine einfache Liste von Buttons keinen professionellen Eindruck hinterlässt.

```
1  //Prüfung auf gültiges Modul
2 Assembly ass = null;
3 ass = Assembly.Load(fileName);
4 if (ass != null){
5         if( (ObjType = ass.GetType(fileName + ".PlugIn")) != null){
6                 //Modul einbinden ...
7         }
8 }
```

Weitere Funktionalitäten des Kerns werden in einer separaten DLL entwickelt und können während der kompletten Durchführung des Projektes erweitert werden. Für den Prototyp werden nur die grundlegenden Klassen betrachtet: Die wichtigste ist eine Datenklasse, welche Konstanten für bestehende Datenhaltungsmöglichkeiten bereitstellt und zur Kommunikation zwischen den Klassen dient (Je nach Komplexität wird eine Trennung notwendig). Desweiteren wird eine Klasse benötigt, um die Zugriffe eines Moduls auf die Datenhaltung zu verwalten. Hier werden die Klassen der Datenhaltung angesprochen, wie z.B. DB oder CSV. Die Entscheidung erfolgt dabei via Analyse zuvor definierter Konstanten.

```
1 if (dataType == Data.DB)
2         db.writeDataLine(dataType, name, data);
3 else if (dataType == Data.CSV)
4         csv.writeDataLine(dataType, name, data);
```

Um eine weitere Funktionalität der Kernschicht zu demonstrieren, wird eine graphische Oberfläche in diese DLL ausgelagert. Eine grundlegende Oberfläche, die oft für Geräte ohne eigene Tastatur genutzt wird, ist eine virtuelle Tastatur. Diese kann unabhängig von Modulfunktionen oder Gerätetypen genutzt werden. Zusätzlich kann durch die Übergabe eines Eingabeformats (Datum, Geld, Barcode, etc.) die komplette Fehlerdiagnose ausgelagert werden,

um damit den eigentlichen Modulcode noch weiter zu minimieren(Diagnose im Prototyp nicht umgesetzt). Die Kommunikation zwischen Modul und virtueller Tastatur erfolgt dabei über die bereits zuvor entwickelte Datenklasse.

```
1 // Datenelement erstellen
2 Data value = new Data();
3 // ggf. mit Eingabeformat initialisieren
4 Tastatur tastatur = new Tastatur(value);
5 tastatur.ShowDialog();
```

Der nächste Schritt ist die Erstellung der Datenhaltungsschicht. Auch dafür wird ein extra Projekt angelegt, welches als Klassenbibliothek compiliert wird. Der Einfachheit halber wurde für den Prototyp ein CSV-Format entwickelt, welches alle Daten nur durch ein Semikolon trennt. Dafür wurde eine Klasse „CSV“ angelegt, die das Interface für die Hardwareschnittstelle implementiert.

```
1 public class CSV : IFace.IDataLayer
2 { ... }
```

Die gleiche Vorgehensweise wie bei der Datenhaltungsschicht wird für die Kommunikationsschicht sowie für die Hardwareschicht angewandt: Eine DLL mit benötigten Klassen erstellen, Interface implementieren und gegebenfalls wiederverwendbaren Code oder graphische Oberflächen in die Kern-DLL auslagern. Da dieser Prototyp nicht auf einem MDE-Gerät implementiert wird, werden auch die beiden Schichten nur als Simulation entwickelt. Die Hardwareschicht simuliert dabei nur einen eingehenden Barcode und Hüllklassen für die anderen Gerätekomponenten, während die Kommunikationsschicht nur Kopien einer Datei macht, anstatt eine Verbindung mit dem PC aufzubauen. Nachdem alle Randbedingungen erstellt wurden, kann mit der eigentlichen Modulprogrammierung begonnen werden. Zur besseren Demonstration der PlugIn-Struktur werden 2 Module entwickelt: Ein Inventurmodul sowie ein Modul zur Wareneingangserfassung. Für das Inventurmodul wird wieder eine Klassenbibliothek (DLL) angelegt und eine Klasse „PlugIn“ erstellt. Diese wird das entsprechende PlugIn-Interface implementieren, um als PlugIn erkannt zu werden. Dafür wird der Name des Moduls festgelegt und innerhalb der „Show“-

Funktion die erste GUI initialisiert. Diese Oberfläche wird eine graphische Eingabemaske für eine Regal- und Filialbezeichnung enthalten. Für Geräte ohne eigene Tastatur kann an dieser Stelle auf die bereits entwickelte virtuelle Tastatur zurückgegriffen werden. Dem Programmierer steht an dieser Stelle die freie Auswahl über die Aufrufform der Tastatur. Für den Prototyp wird ein normaler Button mit der Aufschrift *Tastatur* gewählt, wobei auch eine kleine Grafik mit einer Tastatur möglich wäre. Die nächste zu entwickelnde Oberfläche wird zur Darstellung der aufgenommenen Barcodes genutzt. Während der Initialisierung der GUI können bestehende Daten geladen und angezeigt werden. Weiter Funktionen, wie zum Beispiel Prüfsummenberechnungen werden in dem Kern entwickelt, da sie damit weiteren Modulen zur Verfügung gestellt werden.

```
1 Object[,] oData = DataLayer.loadData(Data.CSV,new String[1]);
2 ...
3 if(Barcode.validateEAN(ean))
4 {...}
```

Das zweite Modul wird mittels einer anderen Vorgehensweise entwickelt. Diese nutzt die Scannereigenschaften modernerer MDE-Geräte, bei denen der Scanner auch ohne Initialisierung aktiv ist. Der eingehende Barcode wird dabei an die aktuelle Cursorposition geschrieben. Das heißt: Hat kein graphisches Element den Focus, welches Text entgegennehmen kann, wird der Barcode verworfen. Wenn für dieses Problem eine Lösung gefunden wird, ergibt sich daraus ein großer Vorteil. Es muss kein Hardwarezugriff entwickelt werden. Eine Lösungsvariante ist die Trennung zwischen einem Scannmodus und einem Editmodus. Während der Scannmodus aktiv ist, kann das Textelement den Focus nicht verlieren. Ändert sich der Wert in diesem Textelement und entspricht dabei einem gültigen Barcodeformat, kann er als neuer Barcode aufgenommen werden. Über den Editmodus wird diese Fixierung beendet und die Werte können bearbeitet oder gelöscht werden.

5
Zusammenfassung

Die Entwicklung der Architektur hat gezeigt, dass mit dieser Vorgehensweise eine weitaus flexiblere Programmstruktur erreicht wird. Im Gegensatz zu anderen Herstellern, die nur universelle Programme anbieten, lässt sich mit dieser Architektur individuelle Software zum gleichen Preis verkaufen, da die Entwicklungszeit der Module sehr gering ist. Sobald alle Schichten implementiert wurden, benötigt ein eingearbeiteter Entwickler lediglich einen Tag pro Modul.
Durch die Programmierung des Prototyps wurde bewiesen, dass sich die Architektur sehr gut bewährt. Alle kritisch markierten Elemente des Utility-Trees, ließen sich problemlos entwickeln und umsetzen. Die hardwarespezifischen Programmteile lassen sich einfach austauschen und ggf. für andere Programme wiederverwenden. Dadurch ergibt sich unter anderem die Möglichkeit, die Barcodeerfassung über ein Handy mit Kamera zu realisieren. Nachdem ein Foto des Barcodes geschossen wurde, wird dies von der Hardwareschicht analysiert und der entsprechende Wert zurückgegeben. Dies ist nur eines von unzähligen Beispielen, die mit der entwickelten Architektur realisiert werden können.

Ausblick: Während der kommenden Entwicklungen wird sich eine Vielzahl von Modulen und Bibliotheken ansammeln. Dafür sollte im Vorfeld ein geeignetes Ablagesystem überlegt werden, da sonst Dateien überschrieben, verloren oder vertauscht werden.
Des weiteren lässt sich die entwickelte Architektur für alle Arten von Handheld-Pcs (PPC , SmartPhone, Tablet PC) nutzen, da sich durch die Hardwareschicht alle individuellen Gerätetreiber verwalten lassen. Die Anwendungsbereiche befinden sich dabei nicht unbedingt im Bereich der Barcodeerfassung, sondern können auch zur Bestellerfassung in Restaurants oder zur Datenerfassung in Kfz-Werkstätten genutzt werden.

6
Anlagen

A
Screenshots

Abbildung A.1: Prototyp

Abbildung A.2: Inventur-Modul

Abbildung A.3: Erfassung des Wareneinganges

B
DVD-Inhalt

Der Bachelorarbeit ist eine DVD beigelegt, die folgenden Inhalt hat:

Ordner \Datei	Beschreibung
Bachelorarbeit_A4.pdf	Text der Bachelorarbeit im A4-Format.
Bachelorarbeit_A5.pdf	Text der Bachelorarbeit im A5-Druckformat (Vorder- und Rückseite).
Prototyp\Quelltext	Projektverzeichnis der Quelltexte
Prototyp\bin	Programmdateien des Prototypen
Prototyp\.NET	Benötigtes .NET-Framework für den Prototypen

C
Abkürzungsverzeichnis

Kürzel	Beschreibung	
.NET	Software-Plattform	**17**
ATAM	Architecture Tradeoff Analysis Methode	**31**
CCD	Charge Coupled Device - optischer Sensor	**1**
CF	Compact Framework	**20**
COM	Communication	**17**
CPU	Central Processing Unit	**24**
CSV	Comma-Separated Values	**18**
DB	Datenbank	**24**
DBMS	Datenbankmanagmentsystem	**18**
DLL	Dynamic Link Library	**34**
GTIN	Global Trade Item Number	**7**
GUI	Graphical User Interface	**16**
LAN	Local Area Network - Netzwerkverbindung	**1**
LCD	Liquid Crystal Display - Flüssigkristallbildschirm	**12**

Kürzel	Beschreibung	
MB	Megabyte	**17**
MDE-Gerät	Handheld-PC zur **m**obilen **D**aten**e**rfassung	**1**
MS	Microsoft	**18**
MSVS	Microsoft Visual Studio	**20**
MVC	Model-View-Control	**18**
OOP	Objektorientierte Programmierung	**18**
POS	Point of sale	**6**
PPC	Pocket-PC	**38**
RFID	Radio Frequency Identification - Identifizierung mit Hilfe von elektromagnetischen Wellen	**8**
SDK	Software Development Kit	**17**
SUN	http://de.sun.com	**17**
USB	Universal Serial Bus	**1**
WinCe	Windows Compact Edition	**17**
WinMobile	Windows Mobile	**17**
WLAN	Wireless Local Area Network - Drahtlose Netzwerkverbindung	**1**
WWS	Warenwirtschaftssystem	**17**
XML	Extensible Markup Language	**18**

D
Literaturverzeichnis

[1] Inventur Fallbeispiel
http://www.eskapeag.de/pdf/Fallbeispiele/Fallbeispiel_Abel_Schaefer.pdf
verfügbar am 26.08.09

[2] Preise Barcodescanner
http://www.kalischgmbh.de/shop/index.php
verfügbar am 26.08.09

[3] Preise Barcodescanner
http://www.sds-office.de/index.php?list=KAT49
verfügbar am 26.08.09

[4] Preise Barcodescanner
http://www.sds-office.de/index.php?list=KAT107
verfügbar am 26.08.09

[5] Barcodeerfassung mittels RFID
http://www.logismarket.de/barcodat/mobiles-datenerfassungsgeraet-barcode-und-rfid/721836494-259361548-p.html
verfügbar am 26.08.09

[6] Barcodedefinitionen
http://en.wikipedia.org/wiki/Barcode
verfügbar am 26.08.09

[7] Barcodeaufbau
http://www.druckerdoktor24.de/lexikon/barcode_aufbau_strichcode.html
verfügbar am 26.08.09

[8] EAN Beschreibung
http://www.activebarcode.de/codes/ean13.html
verfügbar am 26.08.09

[9] GTIN-Aufbau
http://www.gs1-germany.de/content/standards/identifikationssysteme/produkte_gtin/gtin/aufbau/index_ger.html
verfügbar am 26.08.09

[10] Code 2/5 Beschreibung
http://www.activebarcode.de/codes/code25interleaved.html
verfügbar am 26.08.09

[11] Data-Matrix Beschreibung
http://www.activebarcode.de/codes/datamatrix.html
verfügbar am 26.08.09

[12] RFID Definitionen
http://de.wikipedia.org/wiki/Radio_Frequency_Identification
verfügbar am 26.08.09

[13] Vogel, Oliver: Software-Architektur: Grundlagen - Konzepte - Praxis. 2.Auflage 2008, - 556 S.

[14] Posch, Torsten: Basiswissen Softwarearchitektur: Verstehen, entwerfen, wiederverwenden. 2.Auflage 2007, - 349 S.

[15] NordicID morphic, Schnittstellendokumentation
http://nordicid.com/files/1767_NordicID_SDK_MHL_v1.8.pdf
verfügbar am 26.08.09

[16] Nordic ID morphic Produktbeschreibung
http://www.nordicid.de/produkte/nordic-id-morphic.html
verfügbar am 26.08.09

[17] Kommunikation zwischen PC und MDE-Gerät
http://www.etiscan.de/schnittstellen-anbindungen.html
verfügbar am 26.08.09

[18] Handyscan Programmgenerator
http://www.logismarket.de/ip/ids-mobile-barcodeerfassung-mobile-barcodeerfassung-handyscan-8000-c-387821.pdf
verfügbar am 26.08.09

[19] Gernot Starke: Software-Architektur kompakt- angemessen und zielorientiert. 1.Auflage 2009, - 114 S.

[20] J. Bosch: Design and Use of Software Architectures. 1.Auflage 2000, - 354 S.

[21] P. Clement: Documenting Software Architectures: Views and Beyond. 1.Auflage 2003, - 512 S.

[22] Dokumentation der Architektur
http://www.rheinjug.de/videos/gse.lectures.app/Talk.html#ArchitekturDokumentation/30
verfügbar am 26.08.09

E

Erklärung zur selbstständigen Anfertigung

Erklärung:

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

Bearbeitungsort, Datum Unterschrift