Hochschule
Augsburg University of
Applied Sciences

Visualisierung eines RFID-3D-Scanners
Projekt Dokumentation

von

Falk Alexander, Tobias Scholze, Miriam Berschneider, Felix Wagner, Thomas Hipp, Valon Berisha, Michael Stadelmeier, Manuel Feyrer, Maximilian Schrupp, Jennifer Meier

der

Fakultät für Informatik

12. Juli 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Projektübersicht

1.1 Projektbeschreibung

Es soll eine Visualisierungssoftware entwickelt werden, die Tracking und Tracing für logistische Prozesse unterstützt. Hierfür steht der Hochschule Augsburg ein RFID-Gate (ähnlich [1]) zur Verfügung. Anforderungen für die Software sind u. a.:

- Erfassen von RFID-Tags an Waren auf Päckchen die auf einer Palette durch das RFID-Gate gefahren werden
- Auslesen der ID und Erfassung von logistischen Daten
- Korrelation der Daten zu sinnvollen Informationen wie z. B.: Alarmmeldungen, Statistischen Werten, Überwachung des Prozesses
- Visualisierung der Daten in 3- und 2D

Das RFID-Gate wird mit einer einfachen Middleware geliefert, welche in diesem Projekt angewendet, erweitert und durch Ihre Anwendungen evaluiert werden soll. Denkbare Technologien sind z. B.:

- Complex Event Processing
- Mobile Computing Android o.ä. (für GUI)
- Regelmaschinen

Ihre Projektarbeit wird durch Studierende der Fakultät für Wirtschaft unterstützt, die sich mit dem wirtschaftlichen Nutzen Ihrer Anwendungen auseinander setzen werden.

1.2 Anforderungen des "Kunden"

Um Ihr Projekt nachvollziehen zu können, wird von Ihnen ein ablaufähiger Prototyp des Projektergebnisses erwartet. Es sollte sich um eine Ein-Klick-Lösung handeln, die schnell wieder zum Leben erweckt werden kann. Es bietet sich an, ein VirtualBox-Image zu erzeugen in dem der Prototyp eingebettet ist und auf einfache Weise gestartet

werden kann. Der Prototyp ist Teil des oben angesprochenen Datenträgers. Ein solcher Prototyp soll mit geringst möglichen Aufwand installierbar und Vorführbar sein und die besonderen Qualitäten Ihres Projektergebnisses herausstellen. Zu diesem Zweck verfassen Sie bitte zusätzlich eine kurze Beschreibung (z.B. README-Datei) aus der hervorgeht, wie der Prototyp in Betrieb und zu verwenden ist.

1.3 Anforderungen des Teams

Im Rahmen des Kick-Off's kamen verschiedene Ziele und Erwartungen auf, welche wir als Team an das Projekt stellten. Ein oft angesprochener Punkt war, dass es schade wäre, wenn das Ergebnis des Projekts zu einem "Schubladenprodukt" werden würde.

Vor allem was die Projektplanung und den Ablauf der Meetings betraf, wurden klare Wünsche formuliert. So war uns vor allem Wichtig das möglichst alle Teammitglieder in den Meetings anwesend sind und somit jeder in die Prozesse und Planung eingebunden ist. Dadurch wird niemand außen vor gelassen und jeder ist über den aktuellen Stand informiert. Zudem wurde Vorgeschlagen, dass wir abwechselnd im Meeting kleine Ausschnitte aus unserem Arbeitsgebiet vorstellen könnten, damit jeder zumindest einen groben Einblick davon erhält.

Ziel für die Software war, dass sie Mehrwert besitzt und eventuell später von nachfolgenden Projektgruppen weiterentwickelt werden könnte. Daher war unsere Intention, es leicht verständlich, generisch und modular zu implementieren.

1.4 Dokumentation

Die Dokumentation der Projektergebnisse besteht aus drei Teilen:

Der technischen Projektdokumentation Diese beschreibt die die technischen Ergebnisse des Projektes, also die entstandenen Konzepte, Architekturen, Softwareimplementierungen, usw., in einer Art Handbuch für mögliche Kunden.

Der projektarbeitsspezifischen Dokumentation Hier werden die Erfahrungen und Fortschritte der Projektarbeit, aus Sicht des Teams, erläutert.

Die persönliche Dokumentation Jedes Teammitglied beschreibt aus seiner eigenen Sicht das Projekt und die eigene Leistung die im Team erbracht wurde.

2 Evaluierung von Techniken

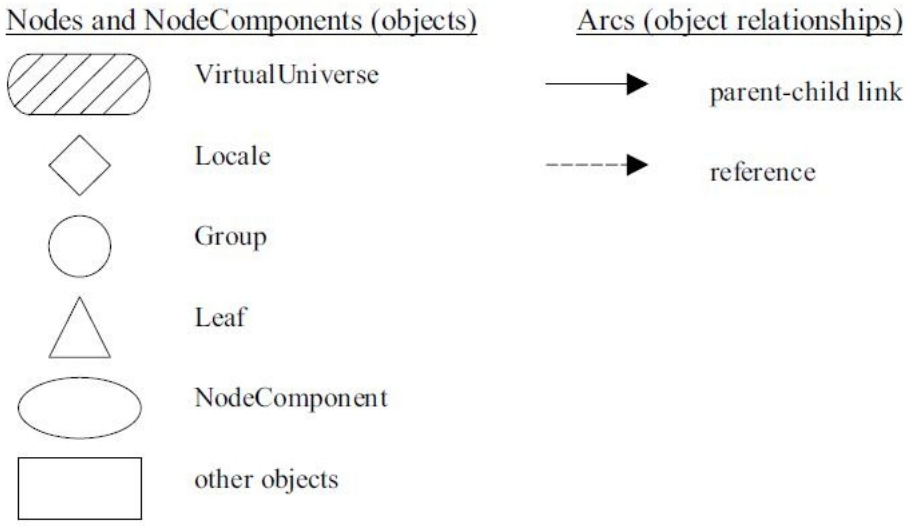
2.1 Java3D

Java3D ist eine Klassenbibliothek zur Erzeugung, Manipulation und Darstellung dreidimensionaler Grafiken. Mit Java3D können dreidimensionale Objekte modelliert, gerendert sowie das Verhalten und die Ansicht gesteuert werden. Java3D kapselt die Funktionalität der zugrundeliegenden OpenGL- oder DirektX-Schnittstelle in ein leicht verständliches objektorientiertes Programmkonzept auf Basis eines Szenengraphen. Im Szenengraph wird der logische Aufbau der darzustellenden Objekte auf eine gleichartig aufgebaute, baumähnliche Struktur abgebildet. Es besteht die Möglichkeit, leistungsfähige 3D-Szenarien zu entwickeln, welche auf der GUI visualisiert werden können.

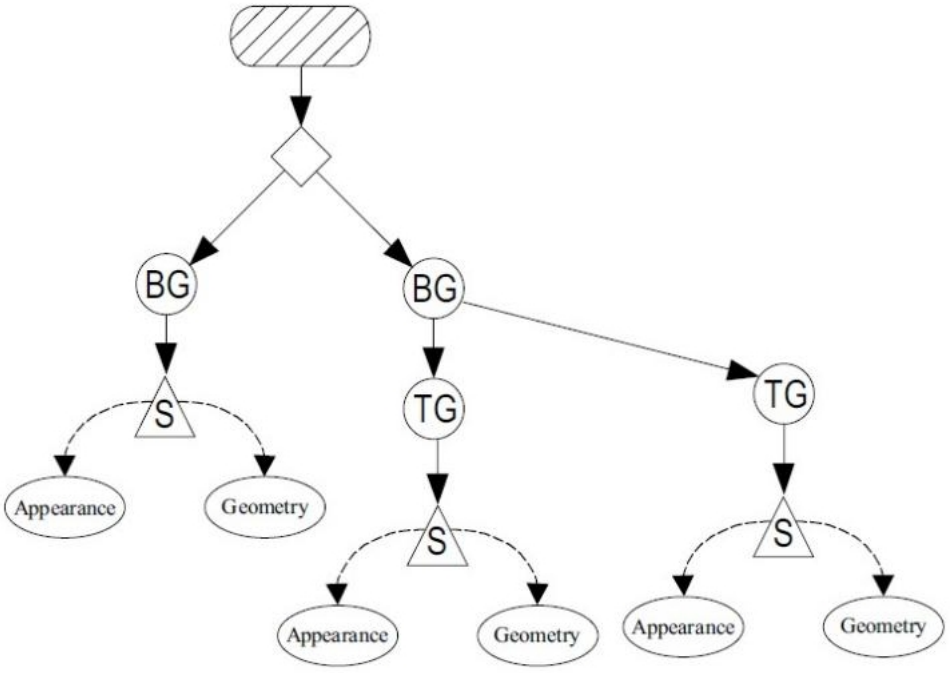
2.1.1 Der Szenengraph

Ein virtuelles Universum in Java3D wird von einem Szenengraphen erstellt. Der Szenengraph wird mithilfe von Java3D-Klasseninstanzen generiert. Der Szenengraph wird also aus Objekten zusammengesetzt, welche die Geometrie eines Objektes, den Ton und das Licht definieren. Man definiert das Erscheinungsbild der optischen und akustischen Objekte. Man kann bei solche einem Szenengraphen auch von einer Datenstruktur von Knoten und Bögen sprechen. Ein Knoten stellt ein Datenelement (eine Java3D-Instanz) dar. Ein Bogen zeigt eine Beziehung zwischen den Elementen. Bögen können zwei Arten von Beziehungen zwischen den Elementen darstellen. Die häufigste Beziehung, welche zur Anwendung kommt ist die Eltern-Kind-Beziehung. Ein Elternknoten kann eine beliebige Anzahl von Kindknoten besitzen. Die andere Beziehung ist eine Referenzbeziehung. Ein Verweis verknüpft ein NodeComponent-Objekt mit einem Szenengraph-Knoten. NodeComponents definieren die Geometrie und das Erscheinungsbild. Abbildung 2.1 zeigt die einzelnen Komponenten eines Szenengraphen. Das nächste Bild (Abbildung 2.2) zeigt ein Beispiel eines Szenengraphen (einzelne Elemente werden nachfolgend beschrieben): Jede Szene hat einen eigenes virtuelles Universum (VirtualUniverse). Dieses wiederum ist im Besitz von Lokalen Objekten (Locale). Ein Locale-Objekt stellt einen Referenzpunkt im virtuellen Universum dar. Dadurch ist die Möglichkeit gegeben, jedes Objekt im virtuellen Universum einem einzigen virtuellen

Abbild 2.1: Szenengraph Komponenten



Abbild 2.2: Beispiel eines Szenengraphen

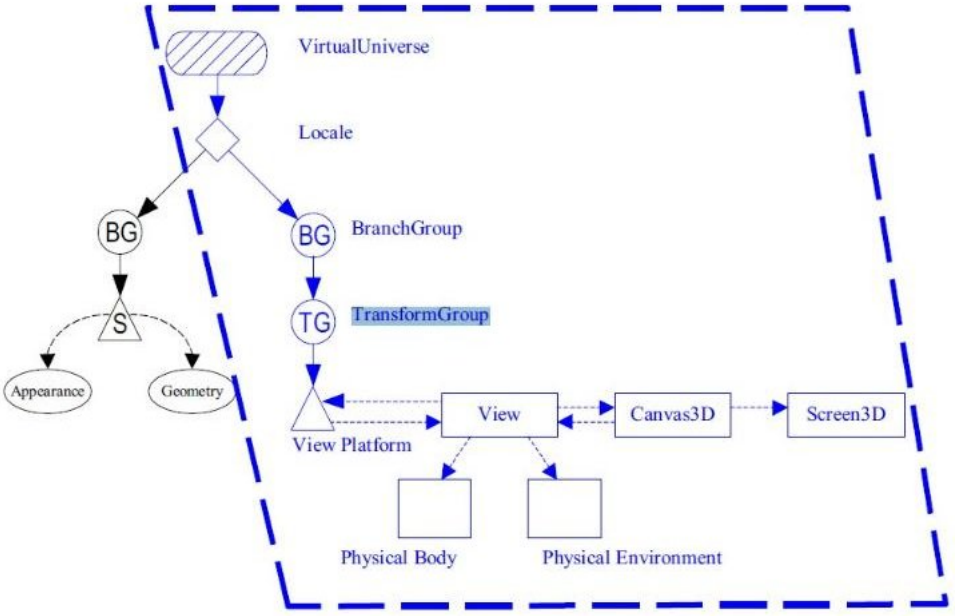


Universum zuzuweisen. Es wäre nämlich auch möglich mehr als ein virtuelles Universum zu haben. Diese können jedoch nicht miteinander kommunizieren. Ferner kann ein Szenengraph-Objekt nicht in mehreren virtuellen Universen gleichzeitig existieren.

Kommen wir aber zurück zum Locale-Objekt. Diesen kann man als Wurzel mehrerer Teilgraphen des Szenengraphen dienen. Im Bild sind zwei Teilgraphen abgebildet, deren

Anfang das *BG* (BranchGroup) bildet. Ein BranchGroup ist die Wurzel eines Teilgraphen. Ferner gibt es zwei Arten von Teilgraphen. Die Abbildung und der Inhalt. Mit dem BranchGroup-Graphen "Inhalt" beschreibt man die Geometrie, das Aussehen, das Verhalten, den Ort, den Ton und das Licht. Der BranchGroup-Graph "Blick" besitzt Paramter, welche die Lage und Richtung bei sich bewegenden Objekten darstellt. Zusammen ergeben sie einen Großteil der Arbeit, welcher der Renderer verrichtet. Ein TG-Knoten (TransformGroup-Objekt) enthält geometrische Transformationen wie z.B. Rotation. Die Aufgaben teilen sich, je tiefer man einen SzenenGraphen-Baum entlang geht. Das nächste Bild (Abbild 2.3) gibt uns einen anderen Blick auf einen Szenengraphen. Das Grundkonzept ist durch das Angebot eines *SimpleUniverse* schon gegeben (blauer Inhalt). Das erstellen eines Objektes SimpleUniverse bietet uns also einen einfachen SzenenGraphen und dessen Inhalte (Knoten) an. Will man trotzdem ein Simple-

Abbild 2.3: Szenengraph eines SimpleUniverse Beispiels



Universe selber erstellen sind bei der Implementierung folgende Schritte zu beachten:

1. Erstellen eines Canvas3D-Objekts
2. Erstellen eines VirtualUniverse
3. Erstellen eines Locale-Objekts
4. Konstruieren des "Blick"-BranchGroup
   (a) Erstellen eines View-Objekts
   (b) Erstellen eines ViewPlatform-Objekts

(c) Erstellen eines PhysicalBody-Objekts

(d) Erstellen eines PhysicalEnvironment-Objekts

(e) Objekte a-d und Canvas3D dem ViewObjekt übergeben.

5. Erstellen des "Inhalt"-BranchGroup
6. Branchgraphen compilieren mit *void compile()*
7. Wenn gewollt kann man nun auch Teilgraphen=Branchgraphen (unter Locale) legen

Nutzt man aber einen SimpleUniverse-Objekt so sieht es folgendermaßen aus:

1. Erstellen eines Canvas3D-Objektes
2. Erstellen eines SimpleUniverse-Objektes, welchem der das zuvor erstellte Canvas3D-Objekt übergeben wird
   - Wenn gewollt kann das SimpleUniverse-Objekt noch weiter den eigenen Vorstellungen angepasst werden
3. Erstellen eines Inhalt"-BranchGroup
4. Compilieren dieses BranchGroup-Objektes mit *void compile()*
5. Einfügen des BranchGroup-Objektes in das Locale-Objekt des SimpleUniverse

2.1.2 Beispiel Implementierung

Ein kleines *helloJava3D*-Code-Beispiel zeigt die Implementierung:

```
/*
 * SimpleUniverse:
 */
public class HelloJava3Da extends Applet {
    public HelloJava3Da() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        add("Center", canvas3D);

        BranchGroup scene = createSceneGraph();
        scene.compile();

        // SimpleUniverse is a Convenience Utility class
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

        // This moves the ViewPlatform back a bit so the
```

```
        // objects in the scene can be viewed.
        simpleU.getViewingPlatform().setNominalViewingTransform();

        simpleU.addBranchGraph(scene);
    }

/*
 * BranchGroup:
 */
    public BranchGroup createSceneGraph() {
        // Create the root of the branch graph
        BranchGroup objRoot = new BranchGroup();
        // Create a simple shape leaf node, add it to the scene graph.
        // ColorCube is a Convenience Utility class
        objRoot.addChild(new ColorCube(0.4));

        return objRoot;
        } // end of createSceneGraph method of HelloJava3Da
} // end of class HelloJava3Da
```

```
/*
 * Main:
 */
public static void main(String[] args) {
    Frame frame = new MainFrame(new HelloJava3Da(), 256, 256);
} // end of main (method of HelloJava3Da)
```

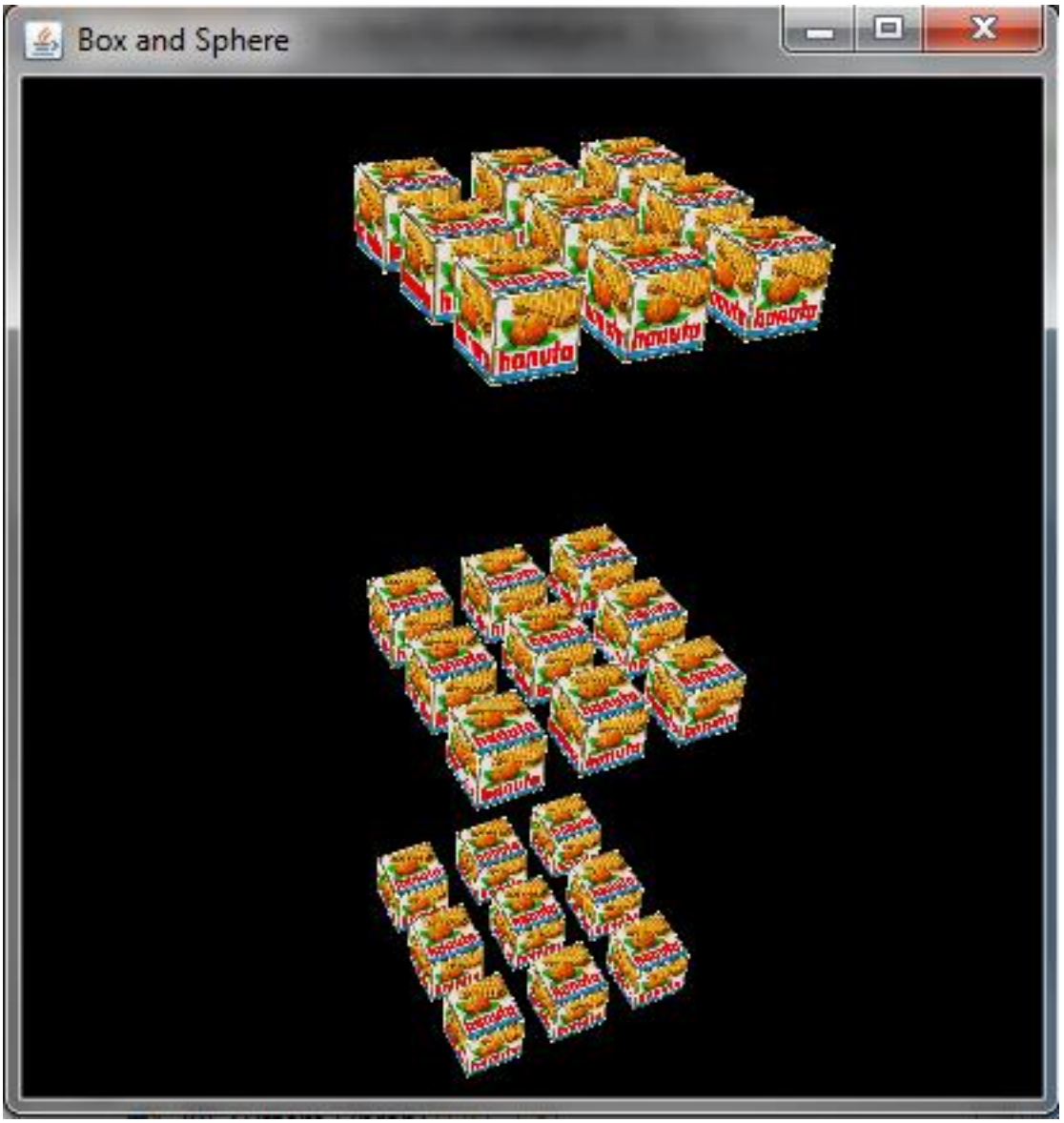
2.1.3 Fazit

Zu Beginn hatten wir uns überlegt die 3D-Visualisierung von gestapelten Boxen mit Swing zu implementieren. Da Java3D jedoch leicht verständlich und gut strukturiert ist um eine schöne 3D-Graphik mit Bewegungen darzustellen, änderten wir unsere Meinung. Leider mussten wir jedoch die Visualisierung in 3D verwerfen, da die Positionsbestimmung von einzelnen Boxen nicht gegeben war. Ein UseCase mit fest definierten Positionen war jedoch möglich. Die Abbildung (Bild 2.4) zeigt Hanuta-Boxen mit fest definierten Positionen.

2.2 CEP

Complex Event Processing (kurz CEP, dt. Verarbeitung komplexer Ereignisse) beschäftigt sich mit der Erkennung, Analyse, Gruppierung und Verarbeitung voneinander abhängiger Ereignisse (engl. Events).

Abbild 2.4: 3D Beispielvisualisierung unserer Tags



2.2.1 Anforderungen

Hoher Durchsatz Je nach Anwendungsgebiet müssen zwischen wenigen hundert und mehreren hunderttausend Nachrichten pro Sekunde verarbeitet werde können

Geringe Latenz Ergebnis-Events sollen zeitnah generiert werden

Komplexe Berechnungen Mechanismen zum Filtern, Aggregieren und Kombination von Events

2.2.2 Relationale Datenbanken vs. CEP

Auf den ersten Blick würden sich für CEP auch relationale Datenbanken anbieten, da die für die Events relevanten Daten meist schon in diesen vorliegen oder in diese anschließend gespeichert werden. Datenbanken sind vorwiegend für manuelles Ausführen von Abfragen optimiert.

2.2.3 Drools Fusion

Drools wurde ursprünglich von Bob McWhirter im Jahre 2001 als open source Business Rules Engine geschaffen. Version 1.0 wurde jedoch niemals veröffentlicht, da der hierfür verwendete Brute- Force-Ansatz eine sehr schlechte Performance aufwies. Stattdessen wurde die Entwicklung an der Version 2.0 gestartet, die auf einer Abwandlung des Rete-Algorithmus basierte. Nachdem Drools im Jahre 2005 in den JBoss Server integriert wurde, wurde es von Grund auf überarbeitet. Neben der Java Implementierung von Drools existiert auch eine .Net Variante.

2.2.3.1 Technische Beschreibung

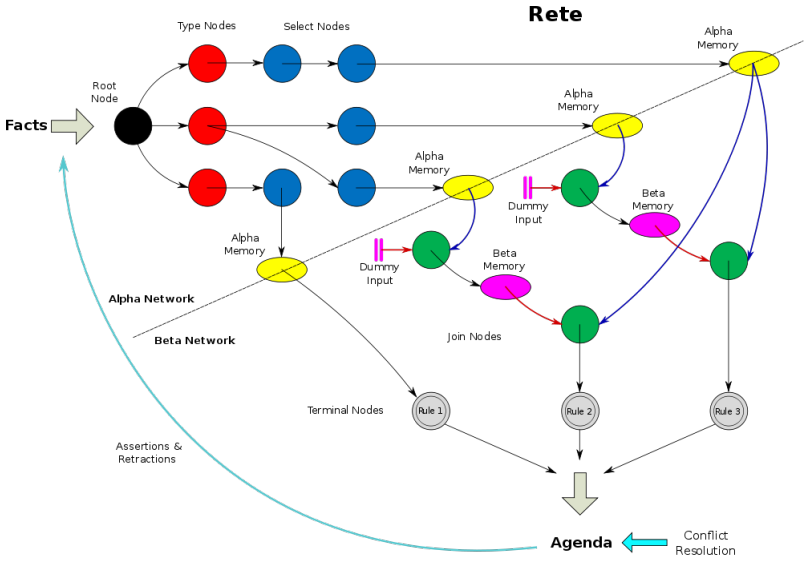
Die Regeln, die Drools auf die Events und Fakten anwendet, werden in einer Knowledgebase abgelegt. Aus dieser Knowledgebase lassen sich KnowledgeSessions erzeugen, die den Kontext der in sie eingefügten Events und Fakten bestimmen. Nach dem Einfügen der Events in die KnowledgeSession werden die Regeln ausgewertet. Basis dieser Auswertung ist der Rete-Algorithmus.

Weiterhin bietet Drools einen Cloud- sowie einen Stream-Modus an. Im Cloud-Modus stehen die Elemente in der Knowledgebase in keinem vorgegeben Verhältnis zueinander und somit müssen stets alle Elemente gegen alle anderen geprüft werden. In dem Stream-Modus sind die Elemente in temporaler Reihenfolge angeordnet, müssen der Engine jedoch auch so übergeben werden. Dieser Modus ist erforderlich, wenn die Engine auf sliding-windows arbeiten soll.

Rete-Algorithmus: Der Algorithmus baut ein Netz von Filtern auf, durch welche die eingehenden Events und Fakten gefiltert werden (siehe Abbildung 2.5).

Die Ausführung des Algorithmus erfordert zwei getrennte Vorgänge. Die rule compilation, während der das Filternetzwerk erzeugt wird, sowie die runtime execution, während der die Elemente durch das Filternetzwerk geleitet werden.

Abbild 2.5: Rete-Algorithmus



Events können der Engine direkt über die API, in Form von Java Objekten, übergeben werden. Ausgehende Events können hierbei über Funktionscallbacks zurückgegeben werden. Falls keine Java Objekte als Datenquelle zur Verfügung stehen, können sogenannte Pipelines genutzt werden. Diese erlauben es beliebige (beispielsweise XML basierte) Datenströme an die Engine anzubinden. Pipelines können hierbei bidirektional agieren. Das heißt sie können Events an die Engine weitergeben und von der Engine erzeugte Events weiterleiten.

2.2.3.2 Syntax

Die Regeln können in Drools in verschiedenen Dialekten erstellt werden. Weiterhin ist es möglich, zusätzliche Dialekte mit Hilfe von Plugins zu erstellen und zu nutzen. So können unter Anderem auch Decision Tables verwendet werden.

2.2.4 Esper

Esper entstand 2004, als Thomas Bernhardt, damals Softwarearchitekt bei einem größeren Finanzinstitut, den Auftrag bekam, mehrere Regel Engines für den Einsatz in einer Monitoring-Anwendung für ein Handelssystem zu evaluieren. Die dabei geprüften Engines erwiesen sich als zu langsam und sperrig für den Einsatz und für die Lizensierung einer CEP-Engine war nicht genügend Kapital vorhanden.

Esper wird in zwei Varianten angeboten, einer als Open Source erhältlichen Basis-Variante und einer erweiterten Ausführung unter kommerzieller Lizenz. Mit EsperHA

(High Availability) existiert auch eine Lösung für hohe Verfügbarkeit. Neben der Java Implementierung existiert auch eine .NET Variante (Nesper).

2.2.4.1 Technische Beschreibung

Esper kehrt das von Datenbanken bekannte Prinzip, Daten zu speichern und dann darauf Abfragen auszuführen, um. Der Anwender speichert zunächst die Abfragen und anschließend prüft Esper einkommende Events auf Übereinstimmungen mit den Abfragen.

Esper unterscheidet zwischen zwei Mechanismen um Events zu verarbeiten. Zum einen wird von Esper eine Event Pattern Language (EPL) angeboten, um Events mit Patterns zu matchen. Zum anderen werden Abfrage-Möglichkeiten auf Eventstreams bereitgestellt. Hierbei werden Fenster, Aggregationen, Joins und Analyse-Funktionen auf Eventstreams unterstützt.

Unterstützte Eingabeformen Esper kann Events in einer Reihe von Darstellungsformen entgegennehmen:

1. von java.lang.Object abgeleitete Klassen, deren Get-Methode den JavaBean- Konventionen entsprechen
2. Maps, hier werden Wertepaare aus Eigenschaftsname und -wert gespeichert
3. XML Dokumentobjektknoten
4. der XML-Streaming API for XML
5. eigenen Klassen durch das EsperIO-Paket (siehe Abschnitt 3.3.3 EsperIO)

2.2.4.2 Verarbeitungsmodell

Das Verarbeitungsmodell von Esper ist kontinuierlich, d.h. Abonnenten von Statements werden umgehend informiert. Abonnenten implementieren das Interface *UpdateListener*. Die vom Programmierer implementierte *update()* Funktion wird von der Engine aufgerufen, sobald Ergebnisse verfügbar sind. Eine andere Möglichkeit, sich über Events informieren zu lassen, stellen die Subscriber-Objekte dar. Die Abfrageergebnisse werden UpdateListenern als EventBean übergeben. Darin enthalten sind die in der Abfrage definierten Eigenschaftswerte. Esper kann in einer Java-Anwendung oder auf einem Application-Server gehostet werden. Beim Zusammenfügen mehrerer Eventstreams zu einem neuen Stream von Events kann auch auf Datenbanken zugegriffen werden, um

weitere Daten anfordern zu können. Neben einer Reihe von vordefinierten Fenstern (z.B. Längen- und Zeitfenster), bietet Esper auch benannte Fenster an. In diese können manuell Events eingefügt oder auch mehrere Eingangsströme zu einem kombiniert werden. Es gibt auch Batchfenster welche eine Reihe von Events (anhand der Zeit oder Anzahl) sammeln und dann nur einmal die update-Methode des UpdateListeners ausführen.

2.2.4.3 Insert- und Remove-Streams

Bei der Betrachtung von Fenstern und Aggregationen sowie deren Verarbeitung durch UpdateListener ist die Unterscheidung zwischen Insert- und Remove-Stream relevant. Der Insert-Stream steht dabei für die neu angekommenen Events, der Remove-Stream für die Events, die das Fenster oder die Aggregation verlassen.

2.2.4.4 Syntax

Abfragen werden mit der EPL (Event Processing Language) formuliert, die stark an SQL angelehnt ist. Hiermit wird deklarativ beschrieben, über welche Events UpdateListener informiert werden sollen. EPL unterstützt auch regular expressions.

2.2.5 Unterschiede zwischen den Engines

Zuerst einige Vor- und Nachteile:

Engine	Vorteile	Nachteile
Drools	Die Regeln sind in einer eigenen Datei und somit übersichtlicher. Ist wesentlich mächtiger und besitzt eine bessere Dokumentation	Stark mit JBoss vezweigt und zu mächtig für das was wir brauchen
Esper	nicht so sehr mit Java verzweigt	kann weniger als Drools und nicht so gute Dokumentation

Drools ist wie Esper ein Open-Source-Projekt, wobei Esper auch eine kommerzielle Variante besitzt, die über erweiterte Funktionalität gegenüber der freien Version verfügt.

Drools und Esper (in der Enterprise Edition) bieten die vorbereitete Möglichkeit der Integration in Business-Rule-Management-Systeme (BRMS).

Der Einstieg in Esper fällt dank der guten, aber dennoch kompakten, Dokumentation leichter als bei Drools. Die Engine eignet sich besonders für relativ simple Regeln. Diese können mit guter Performance bearbeitet werden. Mit komplexeren Regeln wird die Regelerstellung jedoch aufwändiger.

2.3 Hibernate

Hibernate ist ein quelloffenes Javaframework, welches auf Persistenz- und OR-Mapping von Datenstrukturen speziallisiert ist. Hierbei wird eine abstrakte und objektorientierte Sichtweise auf Datenbanken geschaffen, welche es erlaubt unabhängig von der darunter liegenden Datenbank (mySQL, Oracle, postgres), DML-Queries abzusenden. Dies bedeutet im simpelsten Fall, dass der Benutzer mit Datenbanktabellen und Einträgen so umgehen kann, wie aus POJOs - Plain Old Java Object - bekannt. Erweitert betrachtet bietet Hibernate dem Entwickler die Möglichkeit, ohne unnützen, aber immer wiederkehrenden Boilerplate-Code, performante Datenabfragen zu erstellen, ohne sein gewohntes Habitat - die Java Welt - zu verlassen.

2.3.1 Ziele von Hibernate

Ziele von Hibernate zusammengefasst:

- Datenbanktabellen und deren Einträge als Java Objekt bereitstellen
- Abstrahierte Sichtweise auf eine Datenbank
- Entlastet den Entwickler und nimmt unnötige Tipparbeit ab
- Vereinheitlichung von Datenbankinteraktionen durch HQL
- Schutz vor Programmlaufzeitfehlern durch Race-Conditions

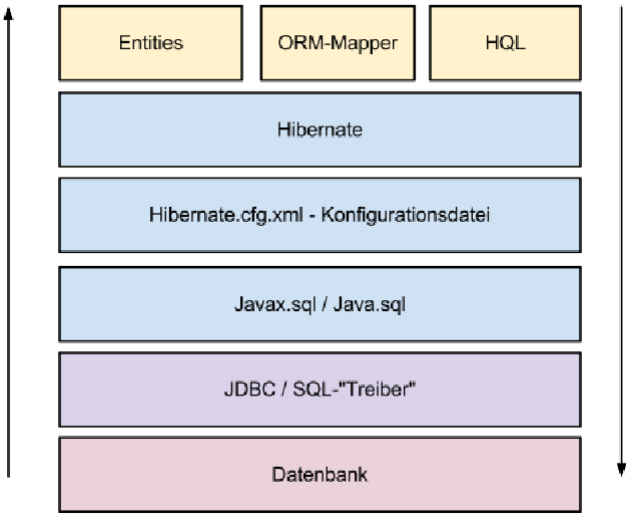
2.3.2 Hibernate-Stack

2.3.2.1 Aufbau

Der Hibernate-Stack setzt auf bereits existierende Java-eigene Schnittstellen auf. Hierbei kann man java.sql und javax.sql als Basis ansehen. Diese Java-Implementationen greifen schließlich via einem JDBC-Treiber auf die darunter liegende Datenbank zu. Dieser Stack wird durch Hibernate in dem Sinne erweitert, dass es die Abstraktion noch weiter

verstärkt. Wenn man dies von oben nach unten betrachtet, bietet dieses Framework eine generische Abfragensprache (HQL). Die Resultate werden anschließend mittels dem ORMapper als Java-Objekte (Entities) repräsentiert. Ein Schaubild dazu befindet sich in Abbildung 2.6.

Abbild 2.6: Aufbau Schaubild



2.3.2.2 Unterstützte Datenbanktypen

Hibernate unterstützt eine Vielzahl von verschiedenen quelloffenen wie auch properitären Datenbanktypen. Neben dateisystembasierten Datenbanksystem stehen schnelle in-Memory Datenbanken zur Verwendung. Hier eine nicht vollständige Liste: MySQL, Oracle 11g, DB2, PostgreSQL, Microsoft SQL Server 2008, SAP DB, Informix, HypersonicSQL, H2 Database, Ingres, Progress, Mckoi SQL, Interbase.

2.4 Maven

Maven ist ein Build-Management-Tool der Apache Software Foundation. Es verwaltet die Programmierung und das Bauen der Anwendung. Maven ist in Java geschrieben und eignet sich deshalb besonders für Projekte, die Java benutzen. Maven steht unter der Apache Lizenz 2.0 und ist von der Free Software Foundation als freie Software anerkannt. Die Hauptziele von Maven sind:

- Vereinfachung des Build-Prozesses
- Bereitstellung eines einfachen Build-Systems

- Vorhalten von qualitativ wichtigen Informationen über das Projekt
- Aufstellung von Richtlinien für eine best practice bei der Softwareentwicklung
- Transparente Aufnahme von neuen Funktionen

2.4.1 Maven Philosophie

2.4.1.1 Convention Over Conguration

Maven verfolgt den Ansatz, dass alle Projekte und Module die gleiche nachvollziehbare Struktur besitzen. Damit zwingt Maven dem Projekt seinen best practice Weg auf. Deshalb sollte sich ein Team möglichst zu Beginn des Projektes für Maven entscheiden. Ein späterer Umstieg kann aufwendig werden. In der Mavenkonvention sind einige Pfade vorgegeben.

```
Sourcecode: $basedir/src/main/java
Ressourcen: $basedir/src/main/resources
Tests: $basedir/src/test
Kompilat: $basedir/target/classes
Pakete: $basedir/target/philosophie
```

2.4.1.2 Einheitliche Herangehensweise

Maven ist in der open source Welt weit verbreitet, dadurch ist jeder Build-Prozess gleich und die Struktur ist ebenfalls einheitlich. Die Einarbeitungszeit in ein neues Projekt wird deutlich reduziert.

2.4.2 Plugin Konzept

Die Grundausstattung von Maven ist spartanisch gehalten. Die Funktionalität steckt in den Plugins, die für Maven zur Verfügung stehen. Werden Komponenten ausgetauscht, z.B. ein Testingframework, so muss nicht das Projekt angepasst werden, sondern nur das Plugin für Maven. Um das neue Framework aufzurufen wird Maven wie gewohnt benutzt. Die Änderungen sind für den Benutzer, aus Maven Sicht, verborgen. Die Handhabung des neuen Frameworks wird im Maven Plugin definiert. Dieses Plugin wird für gewöhnlich von den Entwicklern des Frameworks bereit gestellt. Die Plugins müssen nicht installiert werden. Sie werden in einer Konfigurationsdatei, der *pom.xml*, festgelegt. Deren Download und Installation übernimmt Maven selbst.

2.4.3 Verständnis eines Projekts

Ein Projekt besteht aus mehr als Sourcecode und seinem Kompilat. Es besteht aus einem Team, einer Lizenz und auch Abhängigkeiten zu anderen Projekten. Maven verwaltet den gesamten Lebenszyklus eines Projektes. Es ist nicht nur für das Bauen zuständig. Mit Maven können die Abhängigkeiten zu anderen Projekten aufgelöst werden. Bestehen diese aus weiteren Mavenprojekten, können auch diese Projekte mitgebaut werden. Durch die *pom.xml* stehen anderen Projekten auch Informationen über das eigene Projekt zur Verfügung. So können andere leicht auf die eigene Arbeit aufbauen. Die deskriptive Benutzung der Plugins macht es überflüssig spezische Operationen auf ausgewählten Teilbereichen zu definieren. Die Portabilität zwischen verschiedenen Tools wird durch die *pom.xml* verbessert. Viele IDEs unterstützen dieses Format.

2.4.4 Die *pom.xml*

Die *pom.xml* ist das zentrale Element in jedem Mavenprojekt. Die Abkürzung bedeutet Project Object Model. Daran erkennt man, dass Maven in Projekten denkt, die von einander abhängig sein können. In der *pom.xml* werden alle Informationen abgelegt, die für das Projekt notwendig sind. Einige Beispiele dafür sind der Name des Projekts, die Beschreibungen, die verwendeten Plugins, Kongurationen, die vom Mavenstandard abweichen, sowie die Abhängigkeiten zu anderen Projekten. Jede *pom.xml* erbt seine Standardkonfigurationen von einer Super- POM. Andere Build-Tools wie Ant oder make geben in den Konfigurationsdateien prozedurale Ablaufsbeschreibungen an. In der pom wird deklarativ beschrieben. Es wird festgelegt welche Plugins verwendet werden, wie gepackt werden soll und wo der Quelltext liegt. Ersteres lässt sich auch mit Maven realisieren, entspricht jedoch nicht der Philosophie von Maven Konvention von Konfigurationen.

2.4.4.1 Aufbau

Die pom besteht aus mehreren Abschnitten:

General Projekt Information Grundlegende Informationen über die URL des Projekts, die Lizenz und alle Beitragenden.

Build Settings Weicht das Projekt von den Mavengrundeinstellungen ab, so können hier die Änderungen eingetragen werden z.B. ein anderes Verzeichnis in dem der Quelltext liegt. Ebenso werden hier neue Plugins und Ziele festgelegt.

Build Environment In disem Abschnitt werden Profile für verschiedene Umgebungen abgelegt. In der Entwicklung eines Projekts möchte man einen Entwicklungsserver verwenden. Jedoch, wenn das Projekt produktiv wird, soll ein anderes Profil gelten.

POM Relationship Maven denkt in Projekten. Die Projekte hängen voneinander ab oder bauen aufeinander auf. In diesem Abschnitt werden die Abhängigkeiten und Bestandteile des Projektes definiert. So kann Maven bei Bedarf auch abhängige Projekte bauen.

2.4.4.2 Minimale POM

Das einfachste Beispiel für eine pom ist eine minimale Pom, die keine eigenen Einstellungen besitzt.

```
<project>
    <modelVersion>4.0.0 </modelVersion>
    <groupId>org.foo.bar </groupId>
    <artifactId>my - awesome -app </artifactId>
    <version>1.3.3.7 </version>
</project>
```

Alle vier Angaben sind zwingend notwendig für ein Projekt.

modelVersion Gibt die Versionsnummer des verwendeten Models an. Für Maven 2 3 ist die Versionsnummer 4.0.0 derzeit die einzig zulässige.

groupId Damit wird das Unternehmen, Gruppe oder Institution angegeben, die dieses Projekt erstellt. Dieser Wert ist frei wählbar, sollte jedoch weltweit einzigartig sein. Deshalb lohnt es sich einen Namen zu wählen, der das eigene Unternehmen möglichst genau beschreibt.

artifactId Dieses Feld ist wie groupId frei wählbar und entspricht dem Namen, den das Projekt trägt. Ein Projekt wird durch groupId und artifactId eindeutig identifiziert. Deshalb sollte die Kombination einzigartig sein.

version Dieser Wert muss kein numerischer Wert sein und ist ebenfalls frei wählbar. Werden in Maven Abhängigkeiten definiert, so werden für eine Abhängigkeit genau die letzten drei Felder benötigt.

Alle Informationen die nicht in der minimalen pom stehen, werden von der Super-POM geerbt.

2.4.5 Die effektive POM

Mit dem Befehl *mvn help:effective-pom* wird eine pom erzeugt, die alle Abhängigkeiten, sowohl direkte als auch transitive, besitzt. Es werden auch alle Erbschaften der Super-POM mit einbezogen.

2.4.6 Dependencies

2.4.6.1 Definition von Abhängigkeiten in Maven

Eine Stärke von Maven ist die Auflösung von Abhängigkeiten. Dazu durchsucht Maven alle Repositories, die in der *pom.xml* angegeben sind. Eine Abhängigkeit in Maven wird durch einen Eintrag *dependency* im Abschnitt *dependencies* definiert.

```
<dependencies>
    <dependency>
        <groupId >junit </groupId>
        <artifactId>junit </artifactId>
        <version>3.8.1 </version>
        <scope>test </scope>
    </dependency>
</dependencies>
```

Eine Abhängigkeit zu einem anderen Projekt bedeutet, dass auf Methoden oder Klassen, des anderen Projekts zugegriffen wird. Eine Abhängigkeit wird durch die gleichen Merkmale wie ein Projekt eindeutig identifiziert. Neu hinzugekommen ist der Wert scope. Er gibt an in welchem Teil des Build-Prozesses die Abhängigkeit erfüllt werden muss. Es gibt fünf scopes:

compile Dieser ist der Standard scope falls kein scope angegeben wurde. Compile scopes sind jederzeit im classpath verfügbar und werden mit gepackt.

provided Diese Abhängigkeiten werden vom JDK bereit gestellt und sind beim Kompilieren im classpath vorhanden. Sie sind nicht transitiv und werden nicht mit gepackt.

runtime Diese Abhängigkeiten werden nicht zum Kompilieren benötigt, aber zum Ausführen und Testen.

test Diese Abhängigkeiten werden nur zum Testen benötigt, nicht zum Ausführen der Software.

system Dieser scope sollte vermieden werden. Maven geht davon aus, dass diese Abhängigkeiten vom System bereit gestellt werden. Sie werden in keinem Repository gesucht. Es ist besser ein eigenes Repository einzurichten und dort diese Abhängigkeiten bereit zu stellen, anstatt diesen scope zu verwenden.

3 Projektablauf

3.1 SCRUM

Scrum stellt heute eine der bekanntesten, agilen Methoden dar und das mit gutem Grund: durch seine einfache Struktur und die klar definierten Rollen lassen sich die Scrum-Prinzipien schnell lernen, produktiv einsetzen und so die Vorteile von Agilität schnell ausnutzen.

3.1.1 Grundlegendes

3.1.1.1 Aussagen

- Nicht auf feste "Werte" beharren, sondern auf Individuen eingehen
- Agil in allen Bereichen - Änderungen auch von Kundenseite besser aufnehmbar
- Man nutzt agil die Stärken eines jeden - muss gekonnt sein
- "Scrum ist eine Suchbewegung hin zu dem was der Kunde möchte" (Sprints, usw.)

3.1.1.2 Prinzipien

1. Plan: Konzepte erstellen, Ziele definieren
2. Do: implementieren
3. Check: Schritt zurück und die Arbeit analysieren
4. Act: Auf den Check reagieren

3.1.1.3 Scarf

Model, Social Threats and Rewards

- **S**tatus
- **C**ertainty

- **A**utonomy
- **R**elatedness
- **F**airness

3.1.1.4 Produktives miteinander

Commitment Wir machen, was wir machen wollen

Focus Wir konzentrieren uns auf die Dinge, welche im Commitment beschlossen wurden

Openess Transparenz in Entscheidungen finden und treffen

Respect Jede Arbeit ist gleich "wertvoll". Jeder ist ein Individuum

3.1.1.5 Pull-Prinzip

- Nicht anstacheln oder stressen
- Arbeitsmoral so aufbauen, dass jeder von sich selbst aus das Beste gibt
- Jeder kann hierdurch seine Kreativität spielen lassen. Weniger Arbeit für den Projektleiter

3.1.1.6 Timeboxed

Feste Zeitslots helfen eine wohltuende Struktur herein zu bringen.

Pareto Deadlines und die 80/20 Regel beachten

Flow Machen, anstatt jammern

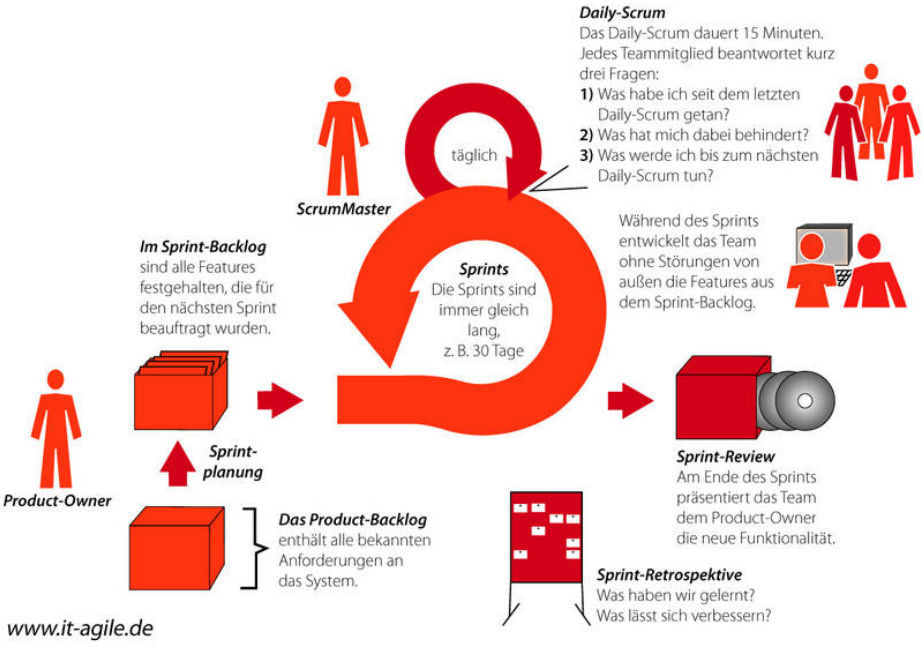
Perfectionism Zusätzlicher Druck es perfekt machen zu müssen.

Fatigue Mentale und körperliche Ausbeutung vermeiden

3.1.1.7 Scrum Übersicht

Siehe Abbild 3.1.

Abbild 3.1: Scrum Ablauf/Übersicht



3.1.2 Scrum Rollen

3.1.2.1 Scrum-Master

- Sorgt dafür, dass das Team produktiv und funktional arbeitet
- Arbeit nicht direkt am Code
- “Gute Fee des Teams”
- Hält den Team den Rücken frei
- Moderator
- Interface von BWL zu Entwickler zu allen anderen

3.1.2.2 Product-Owner

- Kümmert sich um die “Vision” und die Steuerung des Projekts
- Meistens extern (Kunde, Professor)
- Sammelt Anforderungen – Ansprechpartner – “hält den Kopf nach außen hin”
- Arbeitet mit und nicht gegen das Team trotz Wirtschaftshintergrund
- Nimmt das Produkt ab

- Überzeugt das Team vom Projekt und “reißt sie mit”
- Lässt das Produkt wachsen - Sprint-Denken

3.1.2.3 Team

- Optimal 5-9 Mitglieder
- Organisiert sich selbst um die Arbeit gelöst zu bekommen
- Eigene QA
- Arbeitet mit dem Product-Owner
- Will das Projekt realisieren von sich aus
- Gutes Team sind:
  - Profis – auch in Metaeigenschaften
  - Test Driven Development
  - Wissenaustausch – Pair-Programming und -Thinking
  - Feedback – annehmen – anwenden
  - Respekt – Kernarbeitszeit das kein Leerlauf entsteht

3.2 Kick-off

Der Kick-off im Bayrischen Wald stellte quasi das erste Teammeeting dar, bei dem sich die Gelegenheit bot die anderen Teammitglieder persönlich näher kennenzulernen. Durch das umfangreiche Seminarangebot der Erlebnistage Bayrischer Wald bekamen wir eine optimale Einführung in die Team- und Projektarbeit. Indem wir Feedback zu den letzten Projekten sammelten, konnten wir sowohl die aufgetretenen Probleme identifizieren, sowie die positiven Erfahrungen für dieses Projekt übernehmen. Diese Erkenntnisse konnten wir als Verbesserungen in unser jetziges Projekt einfließen lassen. Für unsere zukünftigen Projektmeetings erhielten wir Grundlagen in Moderationstechnik und im richtigen Feedback geben. Bereits im Kick-off zeichneten sich gewisse Rollenverteilungen ab, welche auch im Projekt beibehalten wurden. Als Abschluss des Seminars besprachen wir unsere Ziele und Anforderungen die jeder an das Projekt stellt. Somit lieferte der Kick-off den Grundstock für einen perfekten Projektstart.

3.2.1 Kick-off Plakate

3.2.1.1 Blick auf vergangene Projekte

Positiv	Negativ
Gute teaminterne Kommunikation	Aufgabenstand unklar
Gemeisterte Herausforderungen	Unzuverlässigkeit
Festgelegte Planungs- und Implementierungsintervalle	Planung
Gruppenzusammenhalt	Professor ohne Ahnung und Motivation
Reflexion	Aufgaben wurden zu wenig honoriert
Keine Schuldzuweisung	ungleiche Vorkenntnisse
Schnelles und gutes Ergebnis	mangelndes Feedback des Betreuers
Prototyp	Kein bzw. zu wenige Teamleiter
Gute Meetings	Ungenaue Aufgabenstellung

3.2.1.2 Projektarbeit

- Teamleiter (“Wer macht es und warum?”)
- Aufgabenverteilung bzw. Arbeitspakete
- Zeitdruck (ECTS Punkte)
- Wettbewerb und Konkurrenz
- Gewinn bzw. Nutzen Maximierung
- Absprachen: Team, Professor, Kunde
- Informationsverarbeitung und -weitergabe
- Zusammenarbeit
- Kommunikation
- Kommunikation

3.2.1.3 Checkliste für Teammeetings

Vor und während dem Meeting

- Kommen die richtigen Leute?
- Bearbeiten sie das richtige Thema?
- Haben sie dasselbe Ziel?
- Haben sie genug Zeit?
- Sind alle Teilnehmer gut informiert und vorbereitet?
- Haben sie eine gute Leitung bzw. Moderation?
- Ist Zeit für Austausch von "Nicht-Sitzungsthemen"?

Nach dem Meeting

- Gab es klare Entscheidungen?
- Gab es klare Konsequenzen?
- Gab es klare Arbeitsaufträge?
- Gab es eine klare Terminierung?
- Gab es ein Protokoll?
- Wurden alle Meinungen gehört?
- Kontrolliert jemand die spätere Umsetzung der Beschlüsse?

3.2.1.4 Einstieg in die Moderation

1. Legitimation einholen
2. Visualisierung, Dokumentation (Moderator oder jemand anders)
3. Ziele festlegen und mit dem Team Grundlagen schaffen
4. Zusammenfassen und konkretisieren
5. Kurz halten (Ziel im Auge behalten, Diskussionen zulassen und abbrechen)
6. Am Ende einen Ausblick geben

7. Teilnehmer aktivieren (Stichwort: Redeanteil, Meinungsäußerung)
8. Zeit im Blick haben
9. Haltung sollte neutral und unparteiisch sein

3.2.1.5 Feedback

Sender

- Ich-Botschaften
- Feedback kann positiv oder negativ sein
- Konstruktiv
- Einzel Dialog oder geeigneten Rahmen wählen
- Zeitnah
- Nicht impulsiv
- Sachlich und nicht persönlich
- Fragen ob Feedback erwünscht
- Muss konkret sein

Empfänger

- Entscheiden ob Annehmen oder nicht
- Zuhören, aufnehmen
- Keine Retourkutsche
- Keine Rechtfertigung
- Verständnisfragen
- Feedback darf eingefordert werden
- Verstehen, dass Feedback subjektiv ist

3.2.1.6 Abstraktion und Transfer in den (Projekt-) Alltag

- Grundlagen schaffen
- Stress vermeiden
- Dokumentation (Was? Wie? Wofür?)
- Ressourcen nutzen
- Aufgaben parallelisieren
- Schritt zurück und sich selbst zuschauen
- Zukunftssicher, also Personen unabhängig Dokumentieren
- Aufgaben aufteilen
- Gelerntes und gehörtes Anwenden

3.3 Vorbereitungen

In der Vorbereitungsphase waren wir zunächst mit der Evaluierung von diversen Techniken und Software beschäftigt. Zudem lasen wir uns in veröffentlichte Forschungsarbeiten zu den Themen IEEE und dem Handbuch des Alien Readers ein. Da die Software des Alien Readers in Java implementiert wurde, entschlossen wir uns ebenfalls Java als Programmiersprache einzusetzen. Erste konkrete Vorstellungen zum Ablauf des Projekts kamen dann mit dem Beginn der Planung auf. Zum Beispiel wurden wöchentliche Meetings, die sich grob an die Agile Softwareentwicklung (SCRUM) anlehnen, vereinbart. Als Kollaborationsplattform wurde uns der Teamster von Tspin zur Verfügung gestellt, welchen wir als Betatester erprobten. Wir begannen mit einfachen Tests des Alien RFID-Readers, um uns mit den Grundfunktionen des Gerätes vertraut zu machen. Auch erste Versuche die Positionierung der Tags zu ermitteln wurden durchgeführt. Zudem wurde eine erste einfache GUI angefertigt. Diese konnte vom Scanner erkannte Tags in grün und nicht erkannte in rot darstellen. Des weiteren wurde der Termin für das erste Treffen mit dem Vertretern des Frauenhofer Instituts festgelegt.

3.3.1 Gewonnene Erkenntnisse

3.3.1.1 Alien Reader

Antennen

- Bessere Empfangsmöglichkeiten bieten die Antennen 2 und 3
- Lampen für die Antennen blinken nicht immer
- Antennen sollten einander zugewandt sein
- Einstellungen, zu den Antennen, scheinen ignoriert zu werden

Reichweite Je nach Dämpfung sind Reichweiten bis zu 3 m möglich. Liegen die Tags übereinander werden (meist) nur die äußeren beiden Tags korrekt erkannt.

3.4 Sprint 1: Prototyp einer GUI mit Daten vom Reader

Nachdem wir verschiedene Build-Management-Tools genauer angeschaut hatten, wählten wir schließlich Maven. Dieses hilft Java Programme standardisiert zu erstellen und zu verwalten. Da nun die CEP Schicht von der GUI getrennt werden sollte, mussten wir uns für eine Java-Komponente für Complex-Event-Processing (CEP) entscheiden. In der näheren Auswahl standen Drools und Esper. Letztendlich viel die Entscheidung zugunsten von Esper um die Daten performant Filtern zu können. Für die Benutzeroberfläche entstand ein Konzept zur Umsetzung der 3D Darstellung und einem simplen Layout Vorschlag. Für das zweite Treffen mit dem Frauenhofer Institut wurde eine Präsentation mit unseren Fortschritten, eingesetzten Techniken und Fragen vorbereitet.

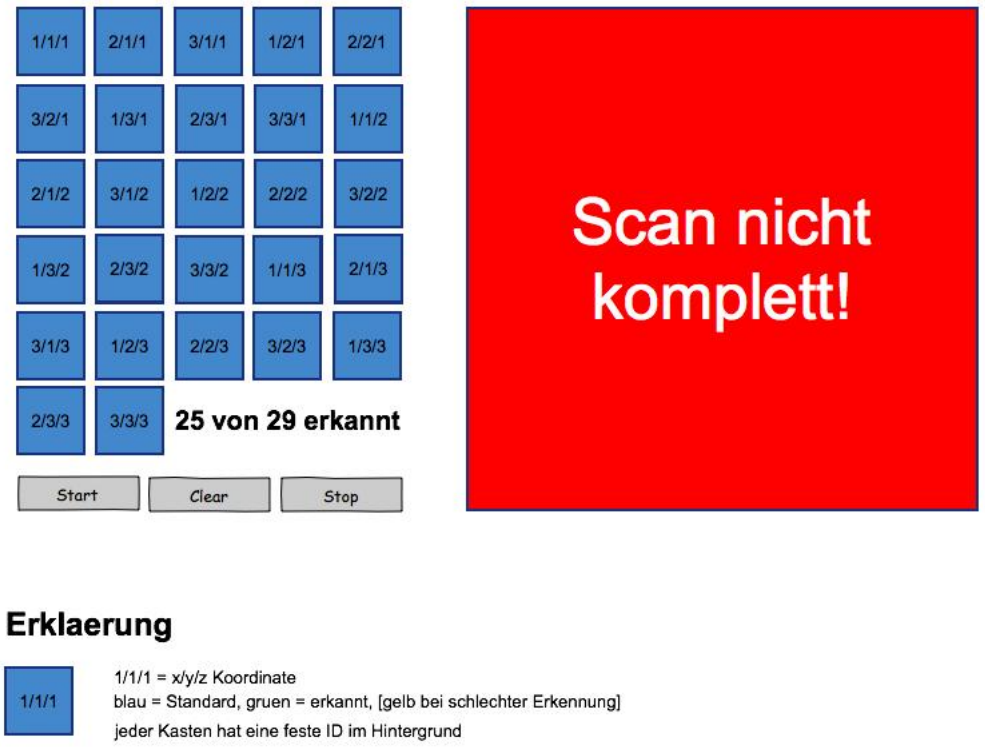
3.4.1 Konzepte

3.4.1.1 1. Vorschlag

Es soll eine simple Visualisierung von einer 5x5 aufgestapelten Produktmatrix widergespiegelt werden. Für eine schnelle Übersicht, ob alle Tags gefunden worden sind, dient eine große rote oder grüne Anzeige rechts neben dem Raster. Diese GUI war nicht für den Endbenutzer gedacht, sondern nur als Hilfsmittel für uns. Im Tag kann neben der Position auch Metainformation (siehe Tag-API-Doku) dargestellt werden. Das Aussehen soll in Abbild 3.2 verdeutlicht werden: Umgesetzt mittels JButtons, JCanvas und JRectangles. Eine explizite Visualisierung mit Java 2D und Java3D wäre hier nicht von Nöten.

Bedenken Durch dieses Layout ist keine 3D Zuordnung mehr möglich.

Abbild 3.2: Veranschaulichung des 1. Versuchs



3.4.1.2 2. Vorschlag

Abbild 3.3 zeigt eine 3D Darstellung unserer Tags Es wäre alles so aufgebaut wie in Vorschlag 1. Jedoch sind die Tag-Kästchen so angeordnet, dass sie ein “echtes” 3D-Modell simulieren. Umgesetzt werden sollte das Ganze mit Java2D und dem TableLayout, oder festen Bildern, welche beschriftet werden.

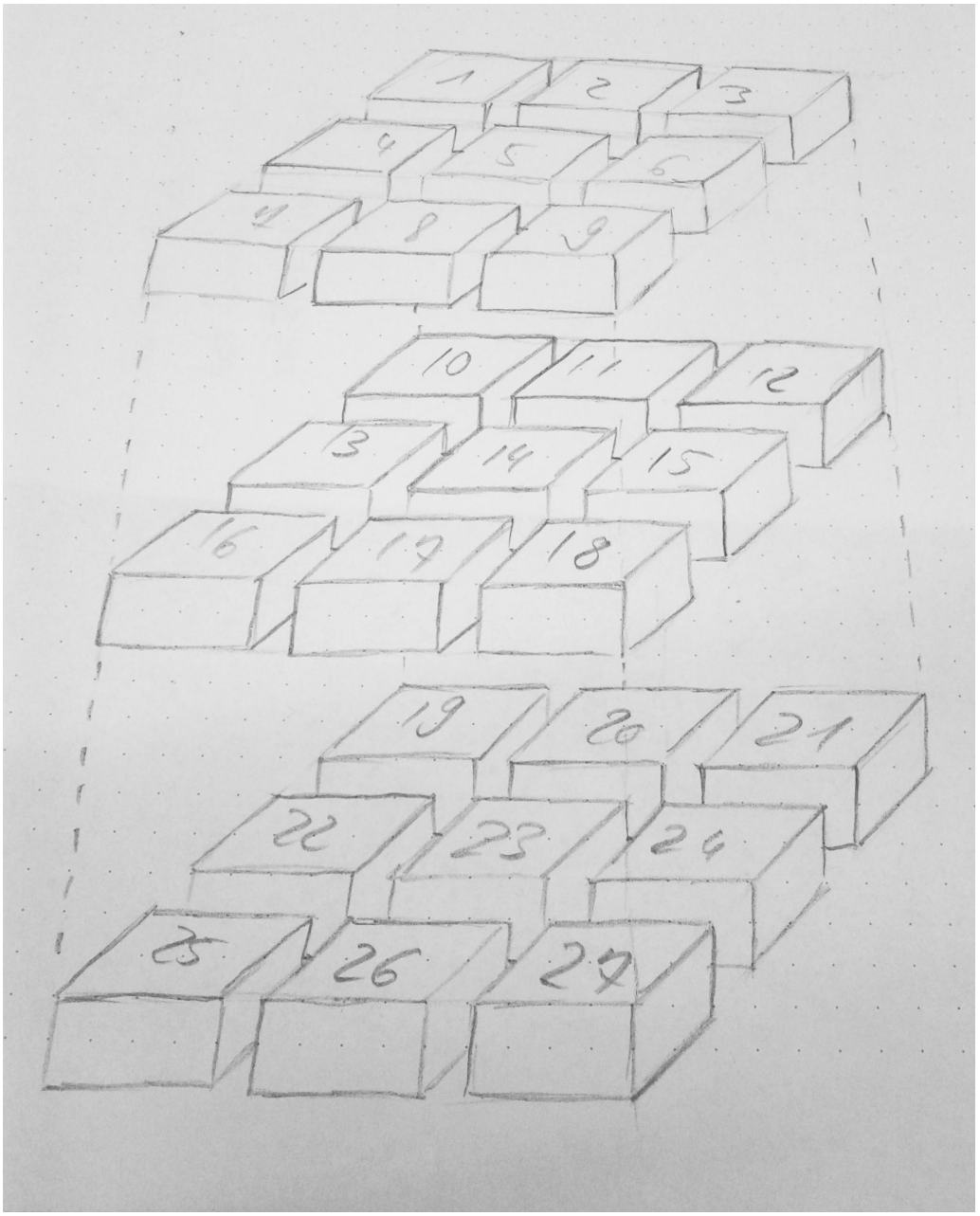
3.4.1.3 Treffen mit Herrn Krupp und Herrn Schöler

Themen Die hier angegebenen Themen sollten spätestens am 02.04. Herrn Krupp und Herrn Schöler übergegeben werden, damit sie sich eventuell darauf vorbereiten können. Wir sollten konkrete Fragen stellen, wie die Zukunft unseres Projekts aussieht. Die Antworten müssen im Meeting festgehalten werden. Anwesende sollten sich Notizen machen.

aktueller Stand

- Was hat jeder Projekteilnehmer bisher gemacht? Was ist seine Agenda?

Abbild 3.3: Veranschaulichung des 2. Versuchs



- Teamster: sind alle zufrieden?
- Rollenverteilung innerhalb des Teams
- Projektbeschwerden? Allein gelassen, gelangweilt?
- Router?

Ziel des Projekts

- Wird es ein Pflichtenheft geben? Haben wir einen Produkt Owner á la Scrum?

- Wie sieht die Verbindung zu dem Fachbereich Wirtschaft aus?
  - Wird es Meetings mit einem Wirtschaftskurs geben?
  - Analysiert der Fachbereich Wirtschaft nur, oder nimmt er Einfluss auf das Projekt?
- Anforderungen müssen klar werden, auch von FISS.

Technische Aspekte

- Gibt es bestimmte Techniken, die zur Anwendung kommen müssen z.B. CEP, SWA?
- Sind wir auf bestimmte Technologien bzw. Programmiersprachen festgelegt? Drools, Esper, Java?
- Ist das Ziel des Projekts eine 2D/3D-Visualiseirung? (Wie die Position bestimmten? API hat weder X, Y, Z noch eine Laufzeit?!)
- Gibt es weitere Anfoderungen als die reine Visualisierung?
- Gibt es architektonische Vorbestimmungen?
  - Native Java Anwendung vs. Webapplikation
  - Monolithische Struktur (Software läuft auf dem PC, an dem auch der Reader angeschlossen ist) vs. Client-Server (Trennung der Komponenten)

Termine

- Wann Treffen mit dem FISS wegen Workshop zu RFID und Co?
- Wann nächstes festes Meeting (mit und ohne Professoren)

3.5 Sprint 2: Refactoring, GUI Konzept, CEP, Versuchsaufbau

Im zweiten Sprint wurde implementiert, dass neue Regeln in der CEP Schicht erzeugt werden können und Esper die Tags an die Benutzeroberfläche weitergibt. Zudem konnte der Reader über die GUI gesteuert werden, d.h. der Scannvorgang konnte gestartet und gestoppt werden. Erste Ideen für eine iOS Oberfläche wurden skizziert.

Da wir beim ermitteln der Tag Positionen keine Erfolge erzielen konnten, waren wir auf das Meeting mit dem Fraunhofer Institut angewiesen, um mit ihnen Erfahrungen auszutauschen. Unter anderem bereiteten wir hierfür einen Fragenkatalog vor:

- Positionierung der Tags?
  - Wie lassen sich die Attribute Direction, Speed, RSSI von einem Alien-Tag ermitteln?
  - Ist es möglich die Bewegungsrichtung von Tags zu bestimmen?
  - Kann man erkennen, ob sich Tags auf eine Antenne zu oder weg bewegen?
  - Welche Materialien (+Dicke) braucht man, um das Feld der Antennen abzuschirmen?
  - Wie breitet sich das Feld der Antennen aus?
  - Ist es möglich durch Abschirmung einen Aufbau zu erhalten, bei dem man sichergehen kann, von welchem Sender ein Tag angesprochen wurde?
  - Wie beurteilt das FIS folgendes Vorgehen:
    * Abschirmung der Sender bzw. Empfänger mit Aluminium beklebten Kartonwänden, so dass wir zwei "Korridore" erhalten, die jeweils aus einem Sender und einem Empfänger bestehen
    * Mitschneiden der gefundenen Tags pro Timestamp in einer Durchfahr-Phase. Beispielsweise 5 Sekunden pro Durchfahrt, 50 Timestamps. Wir schieben n Tags durch den Korridor, der Tag, der als erstes gefunden wird, sollte auch als erstes verloren werden. Der Tag, der als letztes gefunden wird, sollte als letztes verloren werden
    * Durch die Darstellung auf einem Graph kann man so ein "Vorne" und "Hinten" des Wagens bestimmen
    * Durch Hinzunehmen der Daten bzw. des Graphen des zweiten Korridors erhalten wir Fehlergenauigkeit, sowie die Bewegungsrichtung zwischen den beiden Korridoren
    * Eventuell lässt sich durch relativ schmale Korridore wenigstens die Position längs zu den Korridoren auf dem Wagen für jedes Tag bestimmen (werden Tags immer im gleichen Timestamp gefunden? Sind sie in der selben Reihe? o.ä.)
  - Hilft uns eine Bodenplatte beim Auslesen der Tags?
  - Sendestärke? Einfluss, Möglichkeiten?
  - Zeitmessungen, Unterschiede? Wahrscheinlich zu ungenau

  - Gibt es sonstige Störfaktoren oder Leistungs- bzw. Genauigkeitssteigernde Möglichkeiten?
  - Grundsätzlich: sind die Antennen tauglich für unser Vorhaben oder ist deren Sendefeld zu groß?

- Lassen sich 64 Tags (4x4x4) in Kunststoffbehältnissen erkennen?
- Wie beurteilt das FIIS als externe unsere Vorgehensweise und aktuellen Stand
- CEP
  - Welche Techniken wurden schon ausprobiert?
  - Gibt es Erfahrungen mit Esper und/oder Drools?
- Usecases und Szenarien
  - Welche Usecases haben sich ergeben?
  - Welche Usecases wurden bearbeitet und wo gab es Probleme?
- Welche Tag-Klassen haben wir? Was könnte man ändern (EPC, KillPW, AccessPW, etc.)?

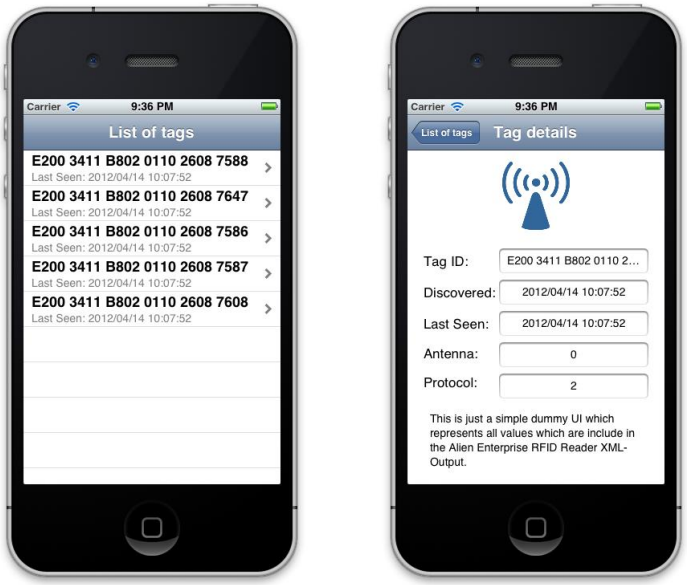
3.5.1 Konzepte

3.5.1.1 iOS Application

In der App soll eine Liste aller gescannten Tags angezeigt werden. Beim Auswählen eines Eintrags sollen Details zu dem jeweiligem Tag angezeigt werden. Die Inhalte basieren auf der XML-Datei welche der Alien Reader sendet. Dieses Verhalten wird in Abbild 3.4 gezeigt.

3.6 Sprint 3: Anfang der neuen GUI, aktives CEP, Einstellungen

Die Umsetzung der GUI Konzepte erfolgte mittels Java Swing sowie Java3D. Im zweiten Treffen mit dem Fraunhofer Institut kristallisierte sich heraus, dass die Ermittlung der Tag Positionen mit diesem Reader-Typ nicht möglich ist. Daraufhin entschlossen wir uns gegen Java3D. Unsere ursprüngliche Projektaufgabe, die bereits vorbereitete 3D Visualisierung der Tags, wurde dadurch hinfällig. Dies stellte eine gravierende Wendung in unserem Projekt dar. Des weiteren entwarfen wir nach dem aufschlussreichen



(a) Listenansicht iOS

(b) Details zu einem Tag auf iOS

Abbild 3.4: Konzeptionsart der iOS Application

Treffen einen neuen Anwendungsfall. Dieser behandelt die Ein- und Auslagerung von Waren. Besonderen Fokus legten wir bei diesem Use Case auf die, in der CEP-Schicht definierten, Regeln, welche Event-Driven triggern. Durch den neuen Anwendungsfall war es erforderlich unsere Hardware mit Lichtschranken zu erweitern. Aufgrund dessen mussten wir unsere Architektur umstellen. Dabei entstanden einige neue Entwürfe zur zukünftigen Architektur unseres Projekts.

3.6.1 Konzepte

3.6.1.1 Anwendungsfälle für die Visualisierung für RFID-Scanner

Motivation dieser Niederschrift Dieses Dokument soll einen Anwendungsfall darstellen, für den die Projektgruppe ss12rfid eine Softwarelösung entwickelt. Das Szenario soll konkrete Features zeigen, die mindestens benötigt werden. Darüber hinaus werden noch wichtige Merkmale der Architektur gefordert. Den Projektteilnehmern von ss12rfid wird damit verdeutlicht welche Anforderungen von außerhalb der Fakultät Informatik an das Projekt gestellt werden.

Rahmenbedingung des Anwendungsfalls In dem System gibt es Personen (durch bestimmte RFID-Tag-IDs identifiziert), verschiedene Waren (jeweils durch bestimmte RFID-Tag-ID-Bereiche identifiziert), sowie ein Lager in dem sich Waren und Personen

aufhalten können. Das Lager wird durch eine Passage betreten. Die Passage besteht aus einem RFID-Tor und zwei Lichtschranken, jeweils eine vor und eine hinter dem Tor.

Formulierung konkreter Anwendungsfälle Im folgenden wird von Anlieferung oder Abholung gesprochen. Das jeweils Fehlende wird impliziert, außer es wird explizit ausgeschlossen.

Durchschreiten der Passage Vom System muss erkannt werden ob eine Person die Passage durchschreitet. Damit ist die Person im Lager bzw. verlässt sie das Lager. Eine Person wird durch eine Tag-ID identifiziert.

Anlieferung von Waren Eine Person liefert Waren an. Dabei durchschreitet eine Person mit Waren die Passage. Im System muss die Verknüpfung zwischen der Person und den Waren entstehen. Beispiel: "Lieferant Müller hat am 01.01.1970 um 13:37Uhr folgende Waren angeliefert: 2 Pakete je 1kg Müllers bestes Mehl und 3 Pakete je 500g fein gemahlene Haselnüsse."

Anlieferung von ungültigen Waren Werden Waren angeliefert, die der Lieferant nicht bereitstellt, so soll die Annahme verweigert werden. Das System soll eine Fehlermeldung bereitstellen, die den Systembetreuer mit einer aussagekräftigen Fehlerbeschreibung informiert und zu einer Lösung auffordert. Beispiel: "Lieferant Müller hat 04.05.12 um 09:53:41Uhr folgende Waren angeliefert: 1 Paket mit 128g Gummibärchen. Diese Waren werden jedoch nicht vom Lieferanten bereitstellt."

Anlieferung einer ungültigen Kombination von Waren Ein Lieferant bringt Waren, die nicht zusammen geliefert werden dürfen.

Historie der Bewegungen Alle Warenveränderungen müssen im System protokolliert werden. Zu jeder Ware muss der Weg, den sie genommen hat, zurückverfolgbar sein. Zu jeder Person muss das Warenportfolio bekannt sein und welche Waren sie wann in und aus dem Lager bewegt hat.

Lagerinhalt Auf Grund der Informationen, die durch die Bewegung der Waren und Personen entstehen, muss über das System abrufbar sein, welche Waren und Personen sich im Lager aufhalten.

Definition von Fehlerfällen

Erkannte Fehlerfälle

Zwei Personen durchschreiten die Passage mit Waren Die Ware kann nicht eindeutig zu einer Person zugeordnet werden. Dieser Fehler kann erkannt, jedoch nicht behoben werden. Es wird eine Fehlermeldung ausgegeben und der Vorgang muss, mit nur einer Person, wiederholt werden.

Waren ohne Person Waren durchschreiten die Passage ohne eine identifizierbare Person (Person ohne Tag-ID). Die Waren werden einer neutralen Personen z.B. der Dummy-Person Unbekannt zugeordnet. Dadurch wird das System etwas robuster gegenüber Personen ohne Tags.

Eine Tag-ID wird doppelt eingelagert Ist eine Tag-ID schon im Lager vorhanden und wird sie dennoch eingelagert, so wird sie ausgelagert. Es liegt die Vermutung nahe, dass die Ware aus dem Lager genommen wurde, ohne die Passage zu durchschreiten.

Person ohne Waren Dieser Fehler wird erkannt und es kann der Person mitgeteilt werden, dass sie sich ohne Waren in der Passage befindet.

Nicht erkannte Fehlerfälle

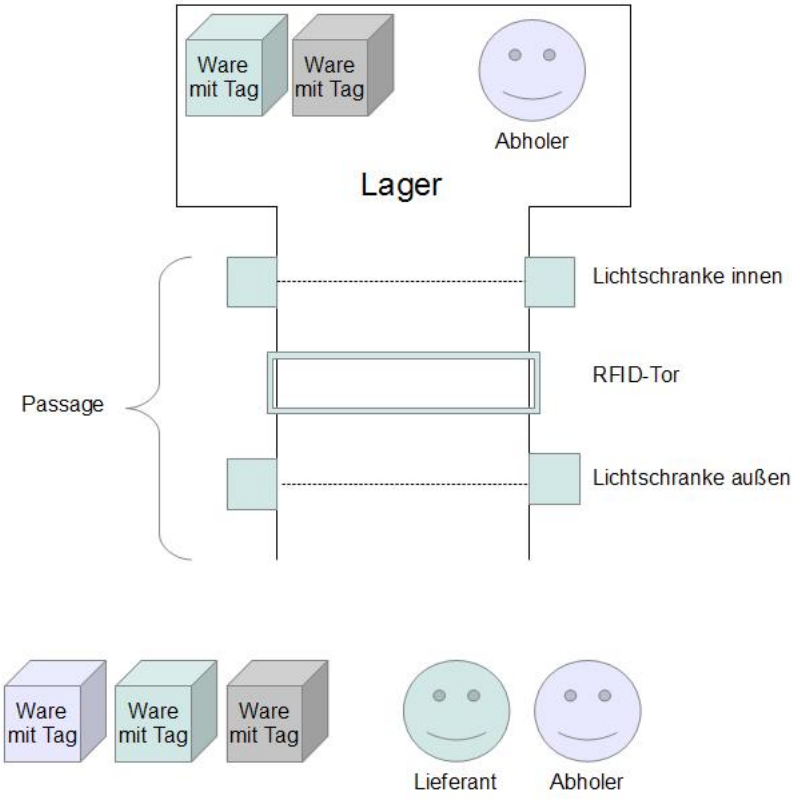
Waren ohne Tag Solche Waren können von diesem System nicht erfasst werden und existieren damit nicht für das System.

Mehrere Paletten in der Passage Sollten sich mehrere Paletten mit Tags in der Passage befinden, kann das System nicht erkennen, welche Tags zu welcher Palette gehören.

Rahmenbedingung

Bewegung Es wird angenommen, dass sich eine Person mit Waren langsam durch die Passage bewegt und nicht für immer stehen bleibt, oder wieder rückwärts aus der Passage heraus läuft. Kein Objekt blockiert die Lichtschranke dauerhaft, das heißt bleibt im Lichtstrahl stehen. Zum besseren Verständnis siehe Abbild 3.5

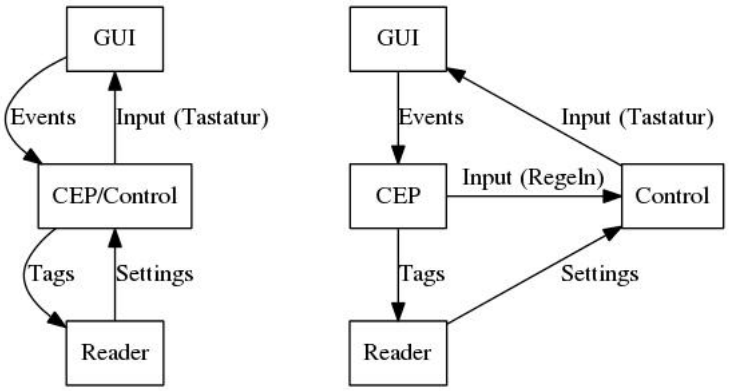
Abbild 3.5: Scrum Ablauf/Übersicht



Lieferung Alle Waren, die zusammen in der Passage sind, werden als eine Lieferung betrachtet. Daraus folgt, das maximal ein Handwagen in der Passage ist.

Einsatz der UI Es sind verschiedene UIs möglich. Alle bis auf die Java GUI sind optional.

Java GUI Es gibt einen zentralen Platz an dem eine Java GUI angezeigt wird. Mit dieser GUI sind viele Informationen abrufbar. Über diese UI können Einstellungen am System vorgenommen werden. Diese GUI muss nicht zwingend aktiv sein, damit die gesamte Anwendung funktioniert. Jedoch sind auf dieser GUI die meisten Informationen abrufbar. Eine weitere GUI befindet sich an der Passage. Im Fehlerfall wird hier ein Warnhinweis eingeblendet und ein Grund für den Fehler wird angezeigt. Ein Beispiel ist “Es dürfen nicht in einer Lieferung Mehl und Nüsse angeliefert werden”. Diese GUI zeigt nur wenige Informationen an. Es können Detailinformationen abgerufen werden. Es können keine bis wenig Einstellungen getätigt werden.



(a) Alte Architektur (b) Neue Architektur ab Sprint 3

Abbild 3.6: Architekturkonzepte

Mobile UI's Über mobile Geräte lassen sich Informationen aus der Anwendung anzeigen. Bei Fehlerfällen soll eine Benachrichtigung angezeigt werden. Es gibt einfache Einstellungsmöglichkeiten. Benachrichtigungen können individuell erstellt werden.

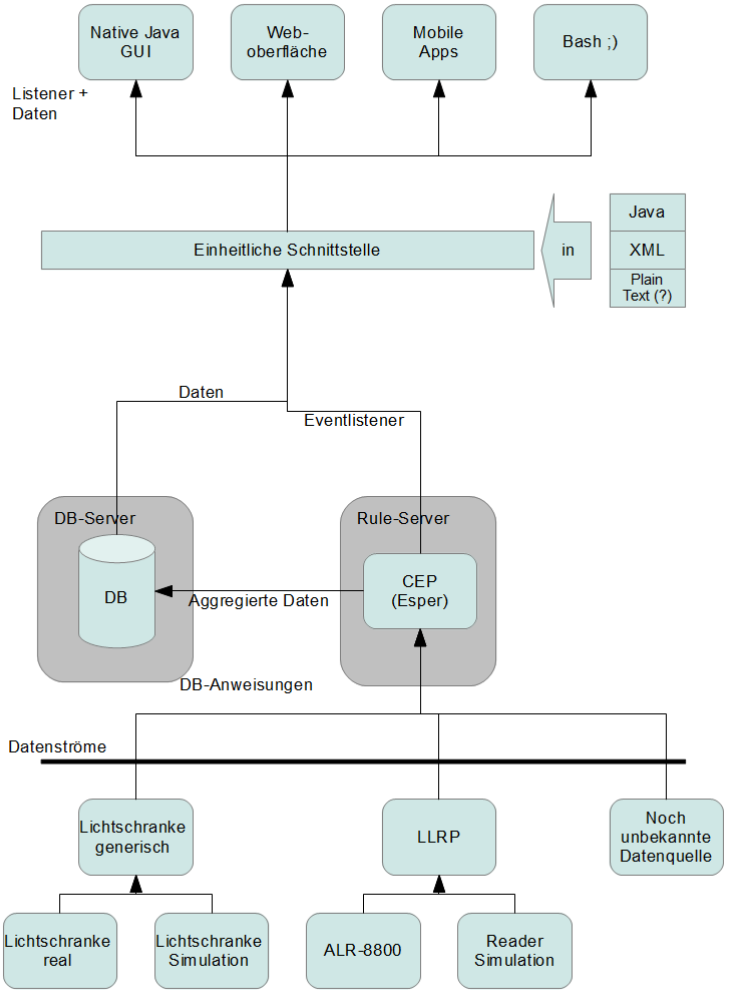
Weboberflächen Plattformen, die Java nicht nutzen können, können über eine Weboberfläche auf das System zugreifen. Die Funktionen sind mit denen der mobilen Oberfläche vergleichbar.

3.6.2 Architekturkonzepte bis zu diesem Sprint

Abbild 3.6 zeigt unsere Implementierung einer Control-Schicht welche nun von der CEP-Schicht getrennt wurde. Control ist nur für die Oberfläche zuständig und sollte von der verarbeitenden Schicht getrennt sein. Damit sollte die Modularität der Software gewahrt bleiben. Die Pfeile sind zu verstehen als "hört auf". Ein Beispiel ist "GUI hört auf Events von CEP/Control". Die Architektur ist stark an das Observer-Pattern angelehnt, da sie eine Erweiterung der GUI darstellt. Daher sind CEP und der Controler-Teil des Observer-Patterns verwoben. Der Reader ist schon durch ein Model abstrahiert. Die GUI und CEP können ohne den Alien-Reader getestet werden. Abbildung 3.7 zeigt wie wir unsere Modularität noch weiter erhöht haben. Alle Benutzerschnittstellen sollten einheitlich auf das Gesamtsystem zugreifen können. Dadurch sollte unsere Software mit jeder Art von UI bedienbar sein. Diese Architektur trägt zurecht den Namen Konzept, da die Umsetzung zu diesem Zeitpunkt nicht abgesprochen war. Die Datenauswertung und -bereitstellung sind verteilt auf einen Rule-Server mit einer Esper-Engine und einem Datenbank-Server. Hier wurde die Anforderung an Firmen bedacht, in denen eigene und mächtige Datenbank-Server eingesetzt werden und auch Esper getrennt vom restlichen System betrieben wird. Die Hardware ist ebenfalls verteilt. Lichtschranken

und RFID-Reader sind eigene Prozesse und müssen über ein Netzwerk mit dem Rule-Server kommunizieren. Für Lichtschranken und Reader sind eigene Simulatoren vorgesehen, um das Konzept testbar zu halten. Die Modularität beschränkt sich nicht nur auf die Austauschbarkeit einzelner Komponenten, sondern wird um die Verteilung im Netzwerk verstärkt. Dieser dezentrale Ansatz erlaubt eine hohe Flexibilität, in Raum und Ressourcen, sowie eine leichte Erweiterung um neue Technologien. Die Pfeile stellen den Informationsfluss dar. Daten fließen von der Hardware zu CEP und von dort aus in die Datenbank und zur GUI. Dieses Konzept wurde nach dem Meeting mit FIIS entwickelt und im darauffolgenden Meeting besprochen. Zur Übersicht wurde aus Bild

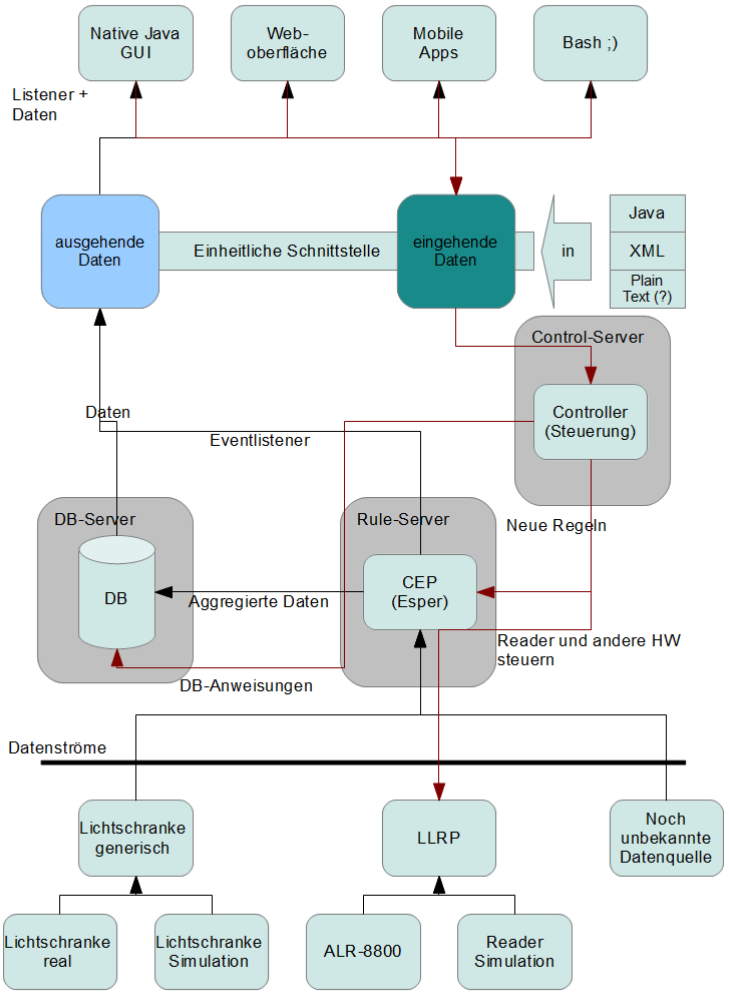
Abbild 3.7: Architektur ohne Steuerung



3.7 die Kommunikation von der GUI abwärts nicht berücksichtigt. Diese ist hier in Abbild 3.8 ergänzt worden. In der GUI werden Einstellungen getroffen, die darunterliegende Komponenten betreffen können. Daher wurde ein Control-Server eingeführt,

der Steuerungsbefehle entgegennimmt und sie in den gewünschten Softwareeinheiten ausführt. Dadurch sollte die Steuerung einheitlich werden. Für einen neuen UI-Typ muss keine eigene Steuerung implementiert werden.

Abbild 3.8: Architektur mit Steuerung



3.7 Sprint 4: DB einbauen, Kommunikation abschließen

Zu Beginn dieses Sprints evaluierten wir verschiedene Lichtschrankenmodelle. Hier lag der Fokus auf Selbstbaukits. Allerdings wurde diese Idee aus verschiedenen Gründen wieder verworfen. Gründe hierfür waren: Zeitnot, Unklarheit über den Arbeitsaufwand zum selbst bauen (Müssen noch Schaltungen gelötet werden?, usw.) und die höheren Kosten entgegen einem bereits montierten Set. Auch was die Datenbankintegration

betraf wurde von uns evaluiert. Es stellte sich heraus, dass Hibernate der aktuelle Standard im Bezug auf Java ist, was ausschlaggebend dafür war es in unserem Projekt einzusetzen. In Esper wurden erste simple Regeln implementiert. Zum Beispiel das gleichzeitige Lesen und Vergleichen aller Tags.

3.7.1 Konzepte

3.7.1.1 Datenbankmodell

Für unser Projekt wurden zwei DB-Modelle von verschiedenen Teammitgliedern entwickelt.

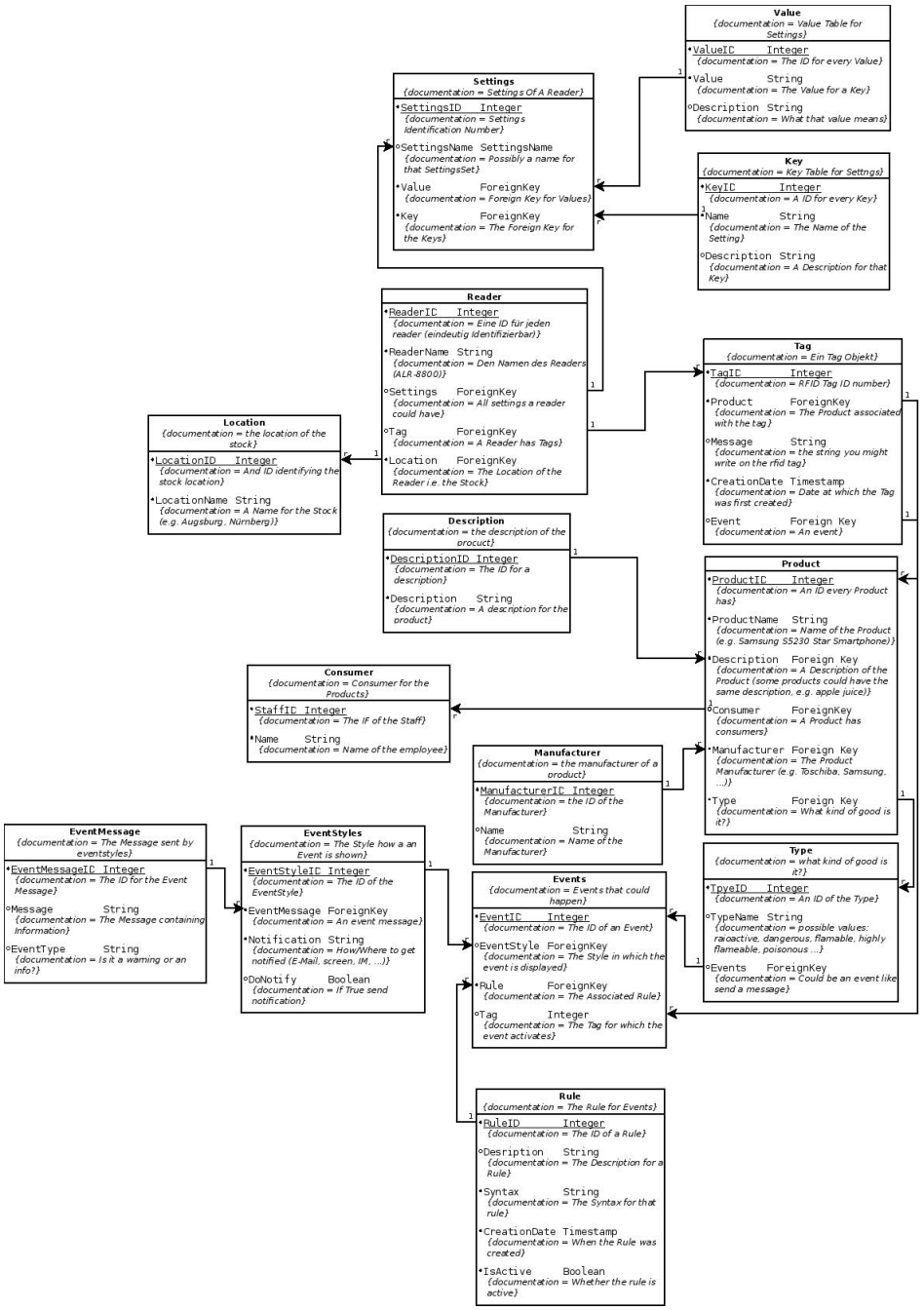
Das fertige Modell sehen wir in Abbildung 3.9.

Überlegungen Wir haben Settings von dem Reader, also wäre es sinnvoll diese ebenfalls in der DB abzuspeichern. Damit haben wir den Vorteil, dass alles zentraler ist (nicht so viele Reader-Settings Dateien) und es hilft uns vor allem, wenn wir in Zukunft mehrere Reader unterstützen wollen. In der Table *Reader* sind mehrere Reader abgespeichert z.B. ALR-8800, ALR-9800 ,... die jeweils *Settings* haben. *Settings* haben ebenfalls einen *Namen*, so könnten wir spezielle Reader Profile erstellen. Die *Settings* haben jeweils *Keys* und *Values*. Diese haben jeweils noch eine *Description*, die bei der Einstellung helfen kann. Dieser Aufbau kann dann auch in der Web-App verwendet werden um den Reader einzustellen. Die Tabelle *Tag* repräsentiert einen RFID Tag, dieser hat eine *ID* (es wäre sinnvoll die ID des Tags zu verwenden, da diese eindeutig sind). Jeder *Tag* ist einem *Produkt* zugeordnet. Ein *Tag* hat auch noch eine *Message* (was auf den Tag geschrieben werden kann). Ein *Tag* hat auch ein *Creation* und *Deletion* Timestamp, um uns zu helfen, ob der Tag bzw. das Produkt noch angeboten wird. Ein *Tag* kann außerdem ein *Event* ausführen bzw. auslösen.

Zuerst gehen wir näher auf das *Produkt* ein. In der Tabelle *Produkt* werden alle unsere Produkte aufgeführt. Ein *Produkt* hat eine *ID*, einen *Namen*, eine *Description*, einen *Type* und einen *Hersteller*. Der *Type* stellt dar was für eine Art von Ware es ist. Der Hersteller oder *Manufacturer* ist selbsterklärend. Zur Tabelle *Location*: Location soll sowas darstellen wie “Kurzlager in Augsburg”. Dabei hat *Location* einen *Namen* (z.B. Augsburg). Die Tabelle *Type* soll darstellen was für ein Produkt wir vor uns haben. Steht vor uns z.B. “Giftmüll” dann soll *TypeName* gleich Giftmüll sein. Die Tabelle hat außerdem auch *Events* welche sich in der Tabelle *Events* wiederspiegeln.

Nun kommen wir zu der Tabelle *Events*. Events können entweder bei Tags kommen oder von der Art des Produkts. Da waren wir uns noch nicht so ganz sicher, ob dies

Abbild 3.9: DB-Modell

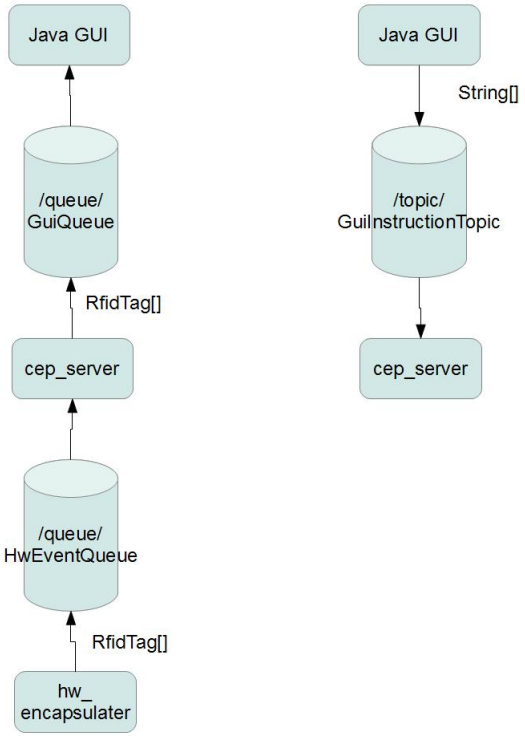


sinnvoll ist. Ein *Event* besteht aus einer *ID*, einem *EventStyle* und einer *Regel*. In der Tabelle *EventStyles* wird definiert, wie ein Event aussieht: z.B. *EventMessage*: stellt sich aus der Message: "Everyone Run away, its the PizzaMonster" und dem Event Type z.B. als "blinkendes Warning Fenster auf dem Screen" dar, *Notification*: E-Mail, Screen und einem Feld DoNotify. Hier dachten wir uns, dass man das DoNotify vielleicht noch ausbaut, sodass eventuell spezielle Personen nochmal ausdrücklich benachrichtigt werden. In der Tabelle *Rules* werden die Drools/Esper Rules verarbeitet bzw. definiert. Hier waren wir uns ebenfalls noch nicht sicher, wie sinnvoll dies ist.

3.7.1.2 Architektur

Abbildung 3.10 zeigt unsere Architektur mit JMS dem Java Message Service.

Abbild 3.10: Architektur mit JMS



3.8 Sprint 5: GUI mit Testdaten, komplexe Regeln

Durch die Implementierung von Hibernate konnte, die schon bestehende Datenbank, an unsere CEP-Schicht angebunden werden. Da die Lieferung der Lichtschranken länger brauchte als gedacht, schrieben wir ein Mock-up um die Use Cases bezüglich der Lichtschranken zu simulieren.

Des weiteren wurden Dummyinformationen für die GUI generiert, welche über eine Weboberfläche dargestellt werden konnten. Dies spiegelte schon im groben unseren Use Case wieder.

Für unsere Anwendungsfälle kamen komplexere Regeln hinzu. Diese definierten unter anderem, dass nach dem Auslösen einer Lichtschranke innerhalb von einer Sekunde ein Tag gelesen werden muss, oder schränkten den gesamten Lesevorgang auf einen bestimmten Zeitraum. Außerdem konnte der zeitliche Ablauf eines Events dargestellt werden.

3.8.1 Konzepte

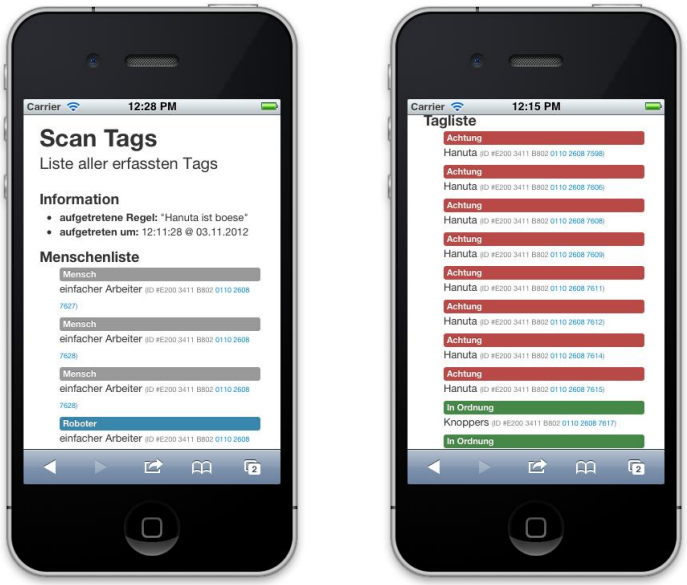
3.8.1.1 Weboberfläche für Desktop oder Tablet

Abbild 3.11 zeigt unsere Weboberfläche im Einsatz. Angezeigt werden eine Liste an RFID-Tags sowie die Arbeiter-Tags und aufgetretene Regeln mit einem Timestamp.

Abbild 3.11: Implementation unserer Weboberfläche für ein Tablet oder einen Desktop-PC

3.8.1.2 Weboberfläche für Smarthpones

Abbild 3.12 zeigt wie die oben implementierte Weboberfläche, auf einem Smartphone aussieht.



(a) Listenansicht iOS

(b) Details zu einem Tag auf iOS

Abbild 3.12: Konzeptionsart der iOS Application

3.9 Sprint 6: GUI Funktionalität, Usecases, LS

Erste Basis Funktionalitäten wurden in das GUI Konzept eingearbeitet. Zusätzlich konnten Einträge aus der Datenbank nun auf der Benutzeroberfläche angezeigt werden. Da endlich die Lichtschranken geliefert wurden, konnten diese eingebunden und die Dummy Implementierung entfernt werden.

In Hinblick auf den Projekttag änderten wir die Online-Beschreibung unseres Projektes in Sempro. Die Anforderungen wurden den nun gültigen Parametern angepasst.

3.10 Sprint 7: Esper Regeln, GUI

Probleme mit Esper führten zu zusätzlichem Arbeitsaufwand, da die Events nicht mehr korrekt triggerten. Unsere GUI wurde mit zusätzlicher Funktionalität ausgestattet. Tags die sich im Lesefeld des Scanners befinden, Infos aus der Datenbank, reinkommende Events sowie qualifizierte Exceptions, auch vom Reader, wurden nun angezeigt. Des weiteren war es nun auch möglich Regeln und Einstellungen auf dem Reader zu setzen.

Da das Projekt langsam dem Ende zuging, begannen die Planungen für den Projekttag. Unter anderem wurde überlegt welche Use Cases wir für unseren Stand umsetzen und welches Equipment hierzu von Nöten ist. Unter anderem wurde auch ein Router mit

eigenem Wlan besorgt, sowie Verhandlungen bezüglich des Standplatzes geführt, da wir für das RFID-Gate relativ viel Platz benötigten. Zusätzlich wurden Konzepte für unsere Plakate ausgearbeitet und in Photoshop umgesetzt. Zur Demonstration des Use Cases wurden Süßigkeiten besorgt, die mit RFID-Tags beklebt wurden.

In der Woche des Projekttages gab es eine Generalprobe für Standaufbau sowie letzte Softwaretests.

3.10.1 Konzepte

3.10.1.1 Regeln und Usecases

Ein Lesevorgang beginnt jeweils mit dem Durchschreiten der ersten Lichtschranke *A* und endet mit dem Durchschreiten der zweiten Lichtschranke *B*. Zwischen diesen beiden Events wird jeder Tag erfasst und an einen *UpdateListener* weitergegeben, welcher die Tags bzw. den Anwendungsfall evaluiert. Sollte binnen acht Sekunden nach dem Unterbrechen von *A*, *B* nicht unterbrochen worden sein, wird alles ignoriert. Für unseren Projekttag haben wir uns folgende Usecases überlegt:

Usecase A Dieser Anwendungsfall wird aktiviert, wenn nach Unterbrechung von *A*, der Tag *E200 3411 B802 0110 2608 7679* gelesen wird. Erlaubt ist die Mitnahme von zwei Artikeln, wobei einer davon Gummibärchen sein muss.

Usecase B Dieser Anwendungsfall wird aktiviert, wenn nach der Unterbrechung von A, der Tag *E200 3411 B802 0110 2608 7680* gelesen wird. Erlaubt ist die Mitnahme von zwei oder vier Artikeln, wobei es sich immer um Paare handeln muss.

Usecase C Dieser Anwendungsfall wird aktiviert, wenn nach Unterbrechung von A, der Tag *E200 3411 B802 0110 2608 7681* gelesen wird. Erlaubt ist die Mitnahme von maximal 3 Artikeln, wobei nur folgende Kombinationen erlaubt sind:

1. Kombination:

- Gummibärchen
- Kinder Country
- Knoppers

2. Kombination:

- Lolly
- Speed
- Ritter Sport

3. Kombination:

- Duplo
- Kinder Country
- Lolly

4. Kombination:

- Knoppers
- Speed

Usecase "Einlagern" Dieser simple Anwendungsfall hat keine Logik implementiert.

Usecase "Auslagern" Dieser Anwendungsfall vertauscht die Lichtschranken-Events, reagiert also bei B gefolgt von A.

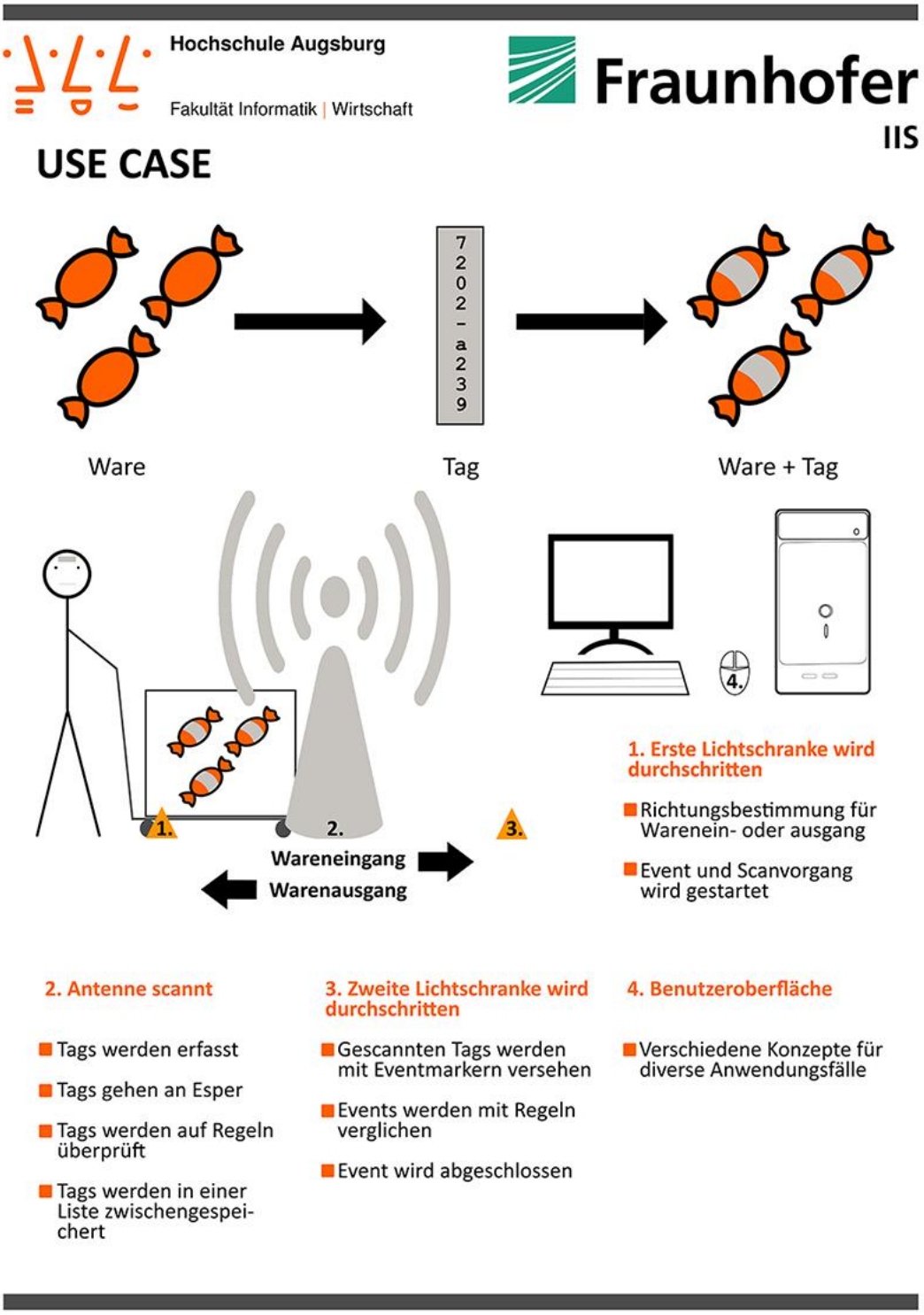
3.10.1.2 Projektplakate

Unser erstes Plakat (siehe Abbildung 3.13) zeigt einen beispielhaften Ablauf eines Wareneinbzw. Warenausgangs.

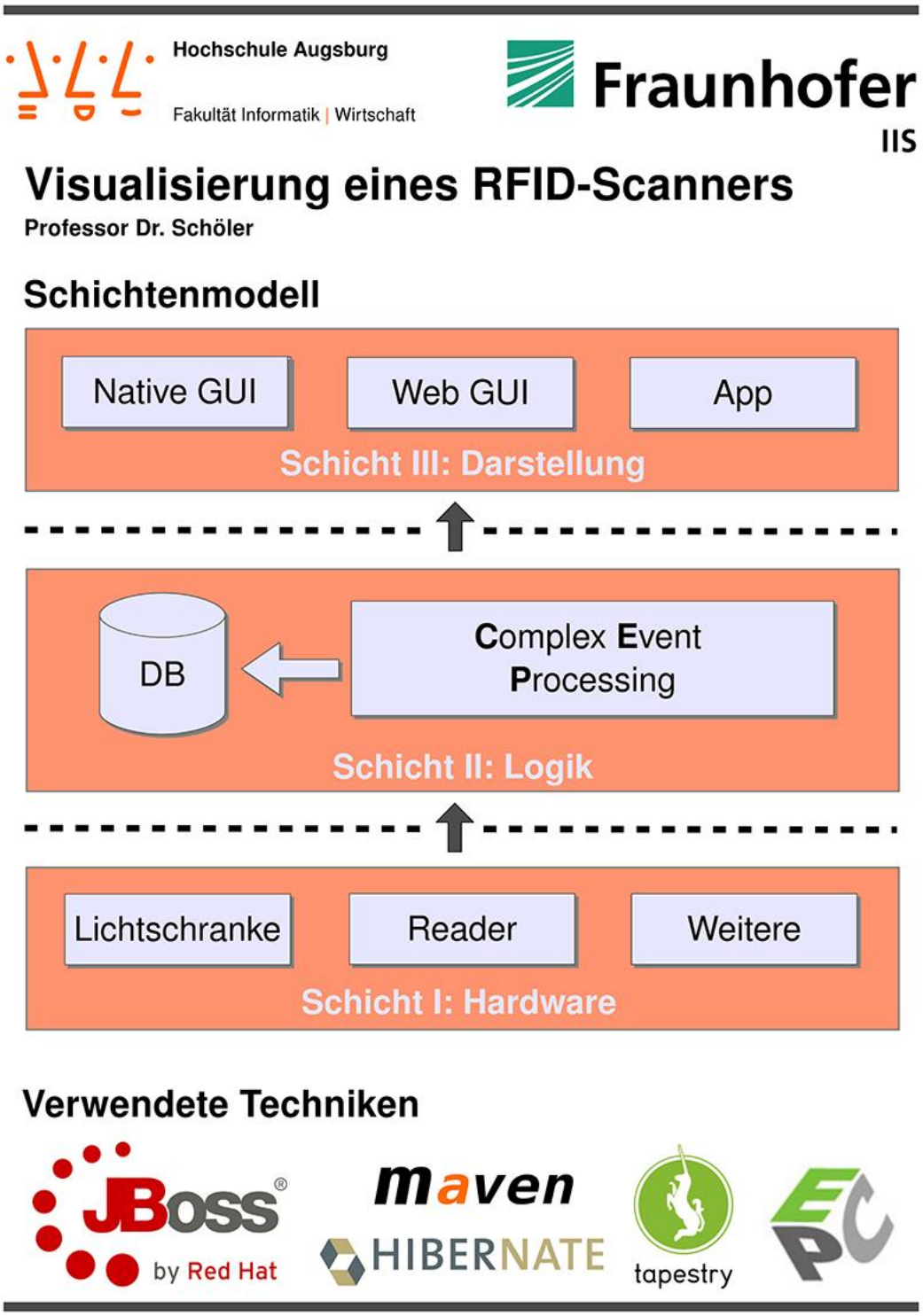
Das 2. Plakat (siehe Abbildung 3.14) zeigt unser eingesetztes Schichtenmodell.

Auf dem 3. Plakat (siehe Abbildung 3.15) sieht man einige Begrifflichkeiten und von uns eingesetzte Verfahren und Techniken.

Abbild 3.13: 1. Plakat



Abbild 3.14: 2. Plakat



Abbild 3.15: 3. Plakat

Bibliography

[1] timo. Rfid gate. URL http://www.flickr.com/photos/timo/151316853/.