HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

ins institute for networked solutions

Presence Engine im Spitalumfeld

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2011

Autoren: Reto Brühwiler und Ricardo Alvarez
Betreuer:Prof. Beat Stettler

Impressum

Kontakt
Alvarez Ricardo, ralvarez@hsr.ch
Brühwiler Reto, rbruehwi@hsr.ch

Druckdatum, 17.6.2011

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Rapperswil, den 17.06.2011

Reto Brühwiler

Ricardo Alvarez

Danksagung

Für die Unterstützung während der Bachelorarbeit möchten wir folgenden Personen einen besonderen Dank ausrichten:

Prof. Beat Stettler für seine Unterstützung und sein wertvolles und konstruktives Feedback.

Marco Facetti für die Betreuung und Unterstützung.

Maurin Egler für die Betreuung und Unterstützung.

Mischa Trecco für die LaTeX Vorlage.

Abstract

Der Spitalalltag besteht für die Angestellten zu einem beträchtlichen Anteil aus Suchen nach freien Geräten, Ärzten oder Patienten. Als Abhilfe werden zunehmend Lokalisierungstechnologien eingesetzt. Allerdings hilft es nicht, ein Gerät zu lokalisieren, das nicht verfügbar ist. Neben dem Standort ist also auch der Zustand des Objektes von Interesse. Bei Menschen sind zusätzlich die Fähigkeiten wichtig, da es beispielsweise wenig nützt, eine freie Person mit den falschen Fähigkeiten zu alarmieren. Von dieser Situation ausgehend war das Ziel, eine flexible State Machine zu programmieren, welche verschiedene Zustände von Personen oder Objekten verwalten kann. Zustandsübergänge sollen von Fremdsystemen automatisch durch Ereignisnachrichten oder manuell ausgelöst werden können. Zusätzlich müssen bei Übergängen von einem Zustand zu einem Anderen auch Aktionen wie z.B. eine Alarmierung ausgeführt werden können. Ausserdem sollte das System nach Inbetriebnahme mit weiteren State Machines oder neuen Aktionen ergänzt werden können.

Zuerst musste eine geeignete Technologie evaluiert werden, die die hohen Anforderungen erfüllen kann. Die State Machine als zentrales Element der Applikation war ausschlaggebend für die Wahl des .Net Frameworks. Da die Applikation hunderttausende von Objekten verwalten können soll, musste zuerst mit Hilfe von Prototypen und Tests nach einer geeigneten Architektur gesucht werden. Um die State Machines flexibel erweitern zu können, werden diese deklarativ in einer Datei beschrieben. Für die Erweiterbarkeit der Aktionen wurde eine Plugin Infrastruktur gewählt. So können im laufenden Betrieb weitere Funktionen hinzugefügt werden.

Die erstellte Applikation besteht aus zwei Teilen. Zum einen die Presence Engine, welche die Verarbeitung der Ereignisnachrichten und Verwaltung der Zustände mit Hilfe der State Machines ausführt. Diese läuft selbständig, verwaltet Objekte und löst Aktionen aus. Zudem wurde eine graphische Oberfläche erstellt, welche die Verwaltung des Systems erlaubt. Dies beinhaltet das Erstellen und Manipulieren neuer State Machine Beschreibungen wie auch das Erfassen und Löschen der Objekte. Zudem werden dem Anwender die aktuellen States der Objekte angezeigt.

Management Summary

Ausgangslage

Der Spitalalltag besteht für die Angestellten zu einem beträchtlichen Anteil aus Suchen nach freien Geräten, Ärzten oder Patienten. Als Abhilfe werden zunehmend Lokalisierungstechnologien eingesetzt. Allerdings hilft es nicht, ein Gerät zu lokalisieren, das nicht verfügbar ist. Neben dem Standort ist also auch der Zustand des Objektes von Interesse. Bei Menschen sind zusätzlich die Fähigkeiten wichtig, da es beispielsweise wenig nützt eine freie Person mit den falschen Fähigkeiten zu alarmieren. Von dieser Situation ausgehend war das Ziel, eine flexible State Machine zu programmieren, welche verschiedene Zustände von Personen oder Objekten verwalten kann. Zustandsübergänge sollen von Fremdsystemen automatisch durch Ereignisnachrichten oder manuell ausgelöst werden können. Zusätzlich müssen bei Übergängen von einem Zustand zu einem Anderen auch Aktionen wie Alarmierung ausgeführt werden können. Ausserdem sollten nach Inbetriebnahme des Systems weitere State Machines und neue Aktionen implementiert werden können.

Vorgehen/Technologien

Zuerst musste die geeignete Technologie für das Projekt gefunden. Die State Machine als zentrales Element der Applikation war ausschlaggebend für die Wahl des .Net Frameworks. Da die Applikation hunderttausende von Objekten verwalten können muss, haben wir mit Hilfe von Prototypen und Tests nach einer geeigneten Architektur gesucht. Um die State Machines flexibel erweitern zu können, werden diese deklarativ in einer Datei beschrieben. Für die Erweiterbarkeit der Aktionen haben wir uns für eine Plugin Infrastruktur entschieden. So können im laufenden Betrieb weitere Funktionen hinzugefügt werden.

Ergebnisse

Die erstellte Applikation besteht aus zwei Teilen. Zum einen die Presence Engine, welche die Verarbeitung der Ereignisnachrichten und Verwaltung der Zustände mit Hilfe der State Machines ausführt. Diese läuft selbständig, verwaltet Objekte und löst Aktionen aus. Zum anderen wurde eine graphische Oberfläche erstellt, welche die Verwaltung des Systems erlaubt. Dies beinhaltet das Erstellen neuer und Manipulieren bestehender State Machine Beschreibungen wie auch das Erfassen und Löschen der Objekte. Zudem werden dem Anwender die aktuellen States der Objekte angezeigt.

Ausblick

Eine mögliche weiterführende Arbeit ist die Weiterentwicklung des graphischen Editors. Als Vorbild hierzu könnte zum Beispiel der Workfloweditor von Windows Workflow Foundation dienen. Autorisierung, Zugriffsberechtigung sowie Schutz vor Manipulationen durch Dritte

und die Sicherung des WCF Services war nicht Teil dieses Projektes. Dadurch bietet der Bereich Sicherheit noch einiges an Weiterentwicklungsmöglichkeiten.

Inhaltsverzeichnis

Teil I

Analyse

1 Problemstellung

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	14.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	14.03.2011	Ricardo Alvarez	Struktur und Einleitung
0.3	15.03.2011	Ricardo Alvarez	Anforderungen

1.1 Überblick

Im Spital werden zunehmend Lokalisierungstechnologien eingesetzt, um Personen und Geräte zu finden. Allerdings hilft es nicht, ein Gerät zu lokalisieren das nicht verfügbar ist. Neben dem Standort ist also auch der Zustand des Objektes von Interesse. Bei Menschen sind zusätzlich die Fähigkeiten wichtig, da es zum Beispiel wenig nützt eine freie Person mit den falschen Fähigkeiten zu alarmieren. Ziel ist es, eine flexible State Machine zu programmieren, welche verschiedene Zustände von Personen oder Objekten verwalten kann. Zustandsübergänge sollen von Fremdsystemen automatisch oder manuell ausgelöst werden können. Beim Übergang von einem Zustand zu einem Anderen müssen auch mehrere Aktionen ausgeführt werden können.

Das folgende Bild gibt einen Überblick.

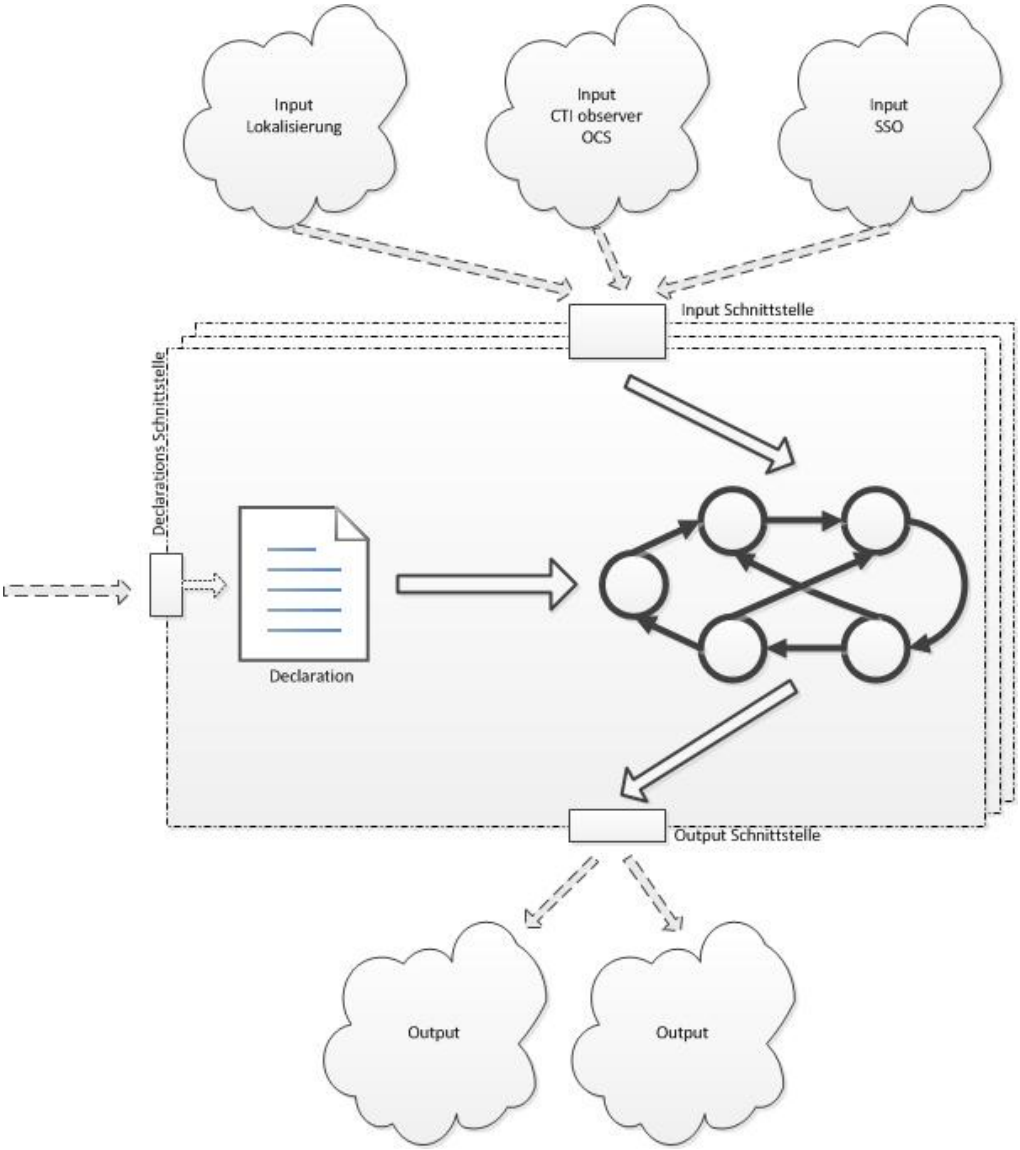


Abbildung (1.1) Big Picture

Kern des Systems bildet die deklarativ beschreibbare State Machine, für Input und Output werden entsprechende Schnittstellen definiert.

Mögliche Lösungsvarianten für diese Vorgaben werden im Rahmen dieser Analyse diskutiert.

1.2 Anforderungen

Aus obigem Überblick lassen sich die folgenden Anforderungen ableiten.

1.2.1 Deklarative Beschreibung

Die State Machines sollen deklarativ beschrieben werden können. Ziel ist es, diese Beschreibung in einer von Menschen lesbaren Form erstellen zu können. Die resultierende Beschreibung soll gespeichert und verändert werden können.

Ideal wäre ein graphischer Editor, um die State Machines zu gestalten und verwalten. Dies würde viel zu Lesbarkeit und Komfort beitragen. Mindestens aber sollte die Beschreibung der State Machine in einer von Menschen lesbaren Sprache, wie zum Beispiel XML, erstellt werden können.

1.2.2 Presence Engine

Die Presence Engine bildet den Kern der Applikation. Sie soll die Deklaration der State Machine einlesen und die entsprechenden statusbehafteten Objekte erstellen können. Sowohl automatische wie auch manuell ausgelöste Statusänderungen sollen unterstützt und die entsprechenden Aktionen ausgeführt werden. Damit dies möglich wird, muss dem Benützer die Beobachtung und Manipulation der State Machines ermöglicht werden.

1.2.3 Aktionen

In einem State soll es möglich sein, automatisch Funktionen auszulösen, zum Beispiel bei Eintritt oder Austritt eines States. Da nicht alle möglichen Funktionen von Anfang an bekannt sind, respektive später Systeme hinzukommen oder wegfallen können, sollen neue Funktionen modular zur Laufzeit hinzugefügt, geändert und gelöscht werden können. Dies muss zur Laufzeit möglich sein, da nicht das gesamte System unterbrochen werden kann, um einen Teilbereich zu ergänzen.

Diese Erweiterbarkeit des Systems ist wichtiger Bestandteil der deklarativen Natur der Presence Engine, so dass auch neue Systeme hinzugefügt werden können.

1.2.4 Schnittstellen

Wie in der Übersicht dargestellt, braucht es an der Systemgrenze eine Schnittstelle, damit Fremdsysteme mittels Nachrichten automatische Statusänderungen auslösen können. Diese Schnittstellen müssen offen genug definiert sein, damit auch zum Implementationszeitpunkt unbekannte Nachrichten verarbeitet werden können.

1.3 Bestehende Technologien

Für die in obigen Anforderungen beschrieben Probleme soll nun die geeignetste Technologie gefunden werden. Dabei wurde die die Evaluation auf Java und Microsoft .Net beschränkt, da das Team mit diesen Technologien die meiste Erfahrung hat.

1.3.1 Deklarative Beschreibung

Es gibt viele standardisierte Sprachen zur Beschreibung von Workflows. Diese eignen sich alle auch zur Beschreibung von State Machines. Als Beispiele zu nennen sind BMNN 2.0, Yawl, BPEL, XAML. Welches Format verwendet wird, hängt direkt von der Technologie für die Presence Engine ab, da diese Systeme meist ein bevorzugtes Format unterstützen. Die Entscheidung wird dementsprechend durch die Wahl der Technologie für die Presence Engine fallen.

1.3.2 Presence Engine

Die Presence Engine bildet den Kern unserer Applikation. Sie muss die verschiedenen State Machines bilden und verwalten. Die für diesen Teilbereich beste Technologie wird ausführlich im nächsten Kapitel evaluiert, da es die Technologie für das ganze Projekt vorgibt.

1.3.3 Aktionen

Die oben erwähnte Modularität wird am einfachsten durch eine Plug-In Infrastruktur erreicht, die es ermöglicht einzelne Module zu verändern oder hinzuzufügen. Da sowohl für Java wie auch Microsoft .Net entsprechende Frameworks zur Verfügung stehen, die dies ermöglichen, ist dies auch kein entscheidendes Kriterium für die Wahl der Technologie.

1.3.4 Schnittstellen

Sowohl in Java wie auch Microsoft .Net gibt es Frameworks, die uns die benötigten Funktionalitäten bieten. So ist auch dies kein entscheidendes Kriterium für die Wahl der Technologie und die entsprechende Detailanalyse folgt nach der Entscheidung.

2 State of the Art

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	15.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	15.03.2011	Ricardo Alvarez	Struktur und Einleitung
0.3	15.03.2011	Reto Brühwiler	Kapitel .Net
0.4	16.03.2011	Reto Brühwiler	Ergänzungen
0.5	17.03.2011	Ricardo Alvarez	Kapitel Fazit
0.6	30.03.2011	Reto Brühwiler	Ergänzungen WF
0.7	06.04.2011	Reto Brühwiler	Ergänzungen WF und bbv
0.8	07.04.2011	Ricardo Alvarez	Performancevergleich WF und bbv

2.1 Einleitung

In diesem Kapitel werden bestehende Technologien evaluiert. Wie im letzten Kapitel aufgezeigt wurde, ist das erstellen und verwalten der State Machines und deren Laufumgebung der bestimmende Teil der Applikation bei der Wahl der Technologie. Der deklarativen Presence Engine am nächsten kommen sogenannte Business Process Management Systeme oder Workflow Management Systeme.

Der Begriff des Business Process Management (BPM) ist ein Begriff aus den Wirtschaftswissenschaften und bezeichnet das Abbilden von Geschäftsprozessen zu deren Identifikation, Gestaltung, Steuerung und Verbesserung. Es werden sowohl technische wie auch generell organisatorische Abläufe beschrieben. Workflow Management heisst der Teilbereich des BPM, der sich auf technisch unterstützte Arbeitsabläufe beschränkt. Die BPM Systeme unterstützen den Benützer beim gestalten der Abläufe, meist durch graphische Editoren, und enthalten eine Workflow Engine, welche die Prozesse abarbeiten und gewünschte Aktionen auslösen.

Eine State Machine kann als Workflow modelliert werden. Allerdings ist Vorsicht geboten, da eine State Machine engeren Regeln folgt und dies bei der Modellierung zwar berücksichtigt werden muss, aber allenfalls nicht durch das System unterstützt wird. Ziel dieser Evaluation ist solche Fragen zu klären, damit die geeignetste Technologie gewählt werden kann.

2.2 Java

In Java gibt es diverse Business Process Management Systeme. Allerdings ist im Open Source Umfeld häufig die Lizenz das Problem, da das Endprodukt auch Open Source sein

muss. Bei den proprietären Lösungen wird viel Wert darauf gelegt, dass das Programm auch von Benutzern ohne technischen Hintergrund zur Verwaltung von Geschäftsprozessen genutzt werden kann. Da nur ein kleines Teilgebiet der Workflow Engine benötigt wird um State Machines abzubilden, dies aber in einem sehr technischen Rahmen, werden diese Programme als ungeeignet erachtet. Im Folgenden werden die evaluierten Workflow-Management-Systeme vorgestellt.

2.2.1 Activiti

Activiti ist eine Java basierte Open Source Business Prozess Management Plattform und bietet neben der technischen Prozessausführung auch Business-IT-Alignement. Es wird unter der Apache-Lizenz veröffentlicht. Dies erlaubt eine Nutzung in proprietären Produkten, was für uns eine Voraussetzung ist.

Zur Abarbeitung der Workflows verwendet Activiti eine Process Virtual Machine. Somit können diverse Spezifikationssprachen zur Beschreibung der Prozesse verwendet werden. Als Standard wird BPMN 2.0 (Business Process Modeling Notation) verwendet, welches durch die Workflow Management Coalition (WfMC) spezifiziert und weiterentwickelt wird.

Ein grosser Teil des Knowhow von Activiti stammt aus der Entwicklung von jBPM Version 1-4, da die Hauptentwickler von jBPM zu Activiti gewechselt sind. Um dies zu Verdeutlichen wurde die erste Version von Activiti anfangs November 2010 als Version 5.0 veröffentlicht.

Activiti besteht aus folgenden 5 Komponenten.

Activiti Engine ist die Java Process Engine und bildet den Kern von Activiti. Wie oben erwähnt ist sie als Process Virtual Machine implementiert. Dies bedeutet, dass in derselben Anwendung verschiedene Spezifikationssprachen verwendet werden können.

Activiti Explorer ermöglicht den Zugriff auf die Activiti Engines. Der Explorer enthält ein Taskmanager, stellt Details zu laufenden Prozessen dar und Reports zu Verfügung.

Activiti Probe hilft Activiti Engines am Laufen zu halten. Der Status jeder Activiti Engine ist hier ersichtlich und allfällige Probleme werden angezeigt. Zudem können in dieser Applikation Log Files eingesehen werden.

Activiti Modeler ist ein Designer, um die BPMN 2.0 Prozesse grafisch zu definieren und bearbeiten.

Activiti Cycle bietet einen allgemeinen Überblick. Der Activiti Cycle wird hauptsächlich verwendet, um die Zusammenarbeit zwischen Stakeholder und Entwickler zu vereinfachen.

Die aktuelle Version 5.3 hat im Test leider noch einige Schwachstellen gezeigt. Unter anderem wurde beim einbinden der Process Engine unter Anleitung eines Tutorials eine NullPointerException geworfen. Dieser Fehler ist bekannt, sollte jedoch laut Bug-Liste

der Entwickler bereits behoben sein. Auch der Designer enthält noch einige kleinere Ungereimtheiten, welche sich aber nur auf die Usability des Designers auswirken und keine Funktionalitätseinbussen zur Folge haben. Die Entwickler haben regelmässig erscheinende, neue Versionen angekündigt, welche hoffentlich diese Kinderkrankheiten ausmerzen werden.

2.2.2 Alternativen zu Activiti

jBPM ist eine flexible Business Process Management Plattform und wird von JBoss entwickelt. Die Spezifikationen und Features von jBPM entsprechen weitgehend denen von Activiti, da der Hauptentwickler zu Activiti gewechselt ist.

Ein entscheidender Unterschied ist, dass jBPM unter der Open Source Lizenz LGPL herausgegeben wird. Diese erlaubt das einbinden von jBPM nur, wenn das Produkt wieder mindestens unter der LGPL Lizenz veröffentlicht oder nur als Bibliothek benutzt wird. Da dies nicht auf unsere Presence Engine zutrifft, erübrigt sich eine genauere Evaluation.

Yawl ist sowohl Beschreibungssprache wie auch Workflow Engine. Speziell an Yawl ist, dass die Entwickler versuchen, alle von der Workflow-Patterns-Initiative definierten Patterns zu unterstützen. Desweiteren basieren Yawl-Graphen auf Petri-Netzen, so dass die formal nachweisbaren Eigenschaften der Petri-Netze genutzt werden können. Deshalb wird Yawl häufig im universitären Umfeld genutzt.

Allerdings ist Yawl, wie jBPM auch, teilweise unter der LGPL Lizenz herausgegeben. Dadurch kann Yawl nicht in diesem Projekt genutzt werden.

Enhydra Shark beinhaltet einen Workflow Editor und Server und nutzt als Sprache, um Workflows zu beschreiben XPDL, welches zu BPMN kompatibel ist.

Gegen die Nutzung von Enhydra Shark spricht wiederum die Lizenzierung, die eine Nutzung in einer proprietären Applikation nicht zulässt.

ActiveVOS ist eine proprietäre Lösung eines Business Process Management Systems und wird von active endpoints vertrieben. Es bietet, wie die Open Source Lösungen, einen Workflow Designer und zusätzlich Unterstützung für WS-Human Task Open Standards, welches spezifisch für die Generierung nicht technischer Prozesse entwickelt wurde. Zusätzlich zum Designer bietet ActiveVOS einen Server als Laufzeitumgebung für die Workflows und eine Konsole für die Verwaltung und Überwachung.

Entscheidend gegen ActiveVOS spricht, dass nicht Workflows, sondern State Machines verwalten werden sollen. Es wird nur ein kleiner Teilbereich benötigt. Dadurch dass es eine proprietäre Software ist und deshalb keine Änderungen vorgenommen werden können, wird es schwierig eine Anpassung an unsere Anforderungen vorzunehmen.

2.3 Microsoft .NET

2.3.1 Windows Workflow Foundation

Windows Workflow Foundation (WF) ist eine auf .NET Framework basierende Klassenbibliothek, welche in der .NET Version 4.0 komplett neu überarbeitet wurde. Durch die Verwendung des .NET Frameworks wird nur der Einsatz in einer Windowsumgebung unterstützt.

WF übernimmt die Erzeugung, Überwachung, Synchronisierung und Terminierung parallel laufender Aktivitäten und eignet sich dadurch besonders für langlebige Workflows. Der automatisch erzeugte Codeanteil kann jedoch sehr schnell riesig und unübersichtlich werden.

Um Ressourcen zu schonen und um Wiederherstellungspunkte für den Fall eines Absturzes zu realisieren, kann der aktuelle Zustand der State Machine in eine SQL Datenbank gespeichert werden.

Mit Hilfe eines, auf Windows Presentation Foundation (WPF) basierenden, Designers können Abläufe grafisch modelliert werden. Dies ist wesentlich überschaubarer und leichter Verständlich als reiner Code. Der Designer lässt sich in eigenen Anwendungen einbinden und anpassen, dies ermöglicht Anpassungen durch den Endanwender. Alternativ können die Abläufe mittels Markup Language realisiert werden, dadurch wird das generieren und editieren mittels externen Tools erleichtert.

Einsatz der Windows Workflow Foundation als State Machine

In der aktuellen Version bietet WF keine eigentliche State Machine oder deren Elemente mehr an. Die Funktionalität der State Machines müssen durch einen Workflow abgebildet werden. Dabei ist ein State kein einzelnes Element, sondern eine logische Gruppierung mehrerer Elemente.

Das zentrale Element in einem solchen konstruierten State stellt das Pick Element dar. Es besteht aus mehreren Branches, pro erreichbaren State wird ein PickBranche definiert. Dieser enthält ein Bookmark Element, welches ermöglicht den Ablauf zu verlassen und bei einem Event an derselben Stelle fortzufahren. In den PickBranches wird definiert, in welchen State gewechselt werden kann. Hierfür wird die Bezeichnung des nächsten States in eine Variable gespeichert.

Das Pick Element wird mit einem Switch Element verbunden, dieses leitet den Fluss entsprechend dem Eintrag der Variable weiter. Vor das Pick Element können die Anweisungen gehängt werden, welche beim Betreten des States ausgeführt werden. Zwischen Pick und Switch entsprechend diejenigen beim Verlassen.

Durch die Zusammenstellung der verschiedenen Elemente wird der Workflow schnell unübersichtlich und komplex.

Verwendete Komponenten

Pick wird verwendet um ereignisgesteuerte Abläufe zu generieren. Das Ereignis bestimmt, welche PickBranch verwendet wird, ähnlich einer Switch Case Anweisung. Bei jedem Durchgang kann dabei nur ein Branch ausgeführt werden.

PickBranch ist ein potenzieller Ausführungspfad innerhalb eines Pick Elements.

Bookmark ist ein Pointer, zu welchem später zurückgekehrt werden kann. Damit ein Idle Modus des Workflows ermöglicht wird muss das Property CanInduceIdle überschrieben werden.

Switch bestimmt den weiteren Fluss in Abhängigkeit eines Wertes. Als Werte können die meisten Standarddatentypen verwendet werden. Es besteht auch die Möglichkeit ein Default Weg zu definieren.

Einbinden des Designers

Als erstes wird eine Instanz der Klasse WorkflowDesigner erstellt. Diese beinhaltet den Designer View und den Property Inspector. Dargestellt wird der Designer in einem Grid, welches aus drei Column besteht.

Grid Aufteilung

Linke spalte Die Toolbox kann eingebunden werden, wobei diese beim Einbinden noch leer ist. Die gewünschten Elemente, welche in verschiedene Kategorien unterteilt werden können, müssen im Code definiert werden. Erstellt werden die Elemente mit Hilfe der ToolboxItemWrapper Klasse. Ziel dieser Wrapper Klasse ist es, die Darstellungsform festzulegen und diese mit einer ToolboxItemInstanz zu verknüpfen. Die erstellten Elemente werden einer Kategorie zugeordnet und diese Kategorie wird in die Toolbox eingebunden.

Mittig befindet sich der eigentliche Designer, in welchem die Workflows mit Objekten der Toolbox erstellt werden können.

Rechte Spalte Der Property Inspector kann eingebunden werden. Dieser ermöglicht es die Eigenschaften der verwendeten Elemente zu verändern.

2.3.2 bbv.Common Hierarchical State Machine

Bbv.Common State Machine ist eine single class State Machine der Firma bbv Software Service AG und wurde unter der Apache Lizenz als Open Source Framework veröffentlicht. Die State Machine beschränkt sich auf die wesentlichen Bestandteile, wodurch sie sehr schnell und simple zu verwenden ist.

Die State Machines lassen sich entweder als Aktive oder Passive State Machine implementieren. Bei der passiven Variante wird die Transition im selben Thread ausgeführt wie der aufrufende Code. Dies bedeutet, dass der aufrufende Code erst fortgesetzt wird, wenn die Transition beendet ist. Bei der aktiven Version wird die Transaktion in einem eigenen Worker-Thread ausgeführt. Somit kann der aufrufende Code weiterfahren, sobald die State Machine den Event in die Queue aufgenommen hat.

Elemente

Transitions können an Bedingungen geknüpft werden, welche Guards genannt werden. Diese entscheiden welche Transaktion verwendet wird, wenn mehrere Transitions für denselben Event deklariert sind.

Actions können beim betreten oder beim Verlassen eines States ausgeführt werden. Die Aktionen beinhalten keine Logik, es werden lediglich Aufrufe externer Methoden hinterlegt.

Fire übergibt ein Event an die State Machine. Die Events werden in eine Queue gespeichert und schnellstmöglich verarbeitet. Für priorisierte Events steht eine FirePriority Funktion zu Verfügung, welche die Events an vorderster Stelle in der Queue platziert.

Logging speichert sämtliche Wechsel der States und ausgeführte Aktionen ab. Die Log Möglichkeit muss zuerst aktiviert werden, durch hinzufügen einer Extension zur State Machine. Die automatischen Logeinträge können durch manuelle erweitert werden. Dem Entwickler stehen hierzu folgende Typen von Logeinträgen zu Verfügung: Debug, Infomation, Warnings, Error, Fatal. Die Logfunktion basiert auf der log4net Library, welche durch Apache aus log4j entwickelt wurde.

Report Es stehen drei verschieden Reportarten zu Verfügung. In der graphischen Form werden die States und Transitions gezeichnet. Des Weiteren könne die Reports als csv File oder als Text ausgegeben werden.

Start / Stopp Die Events werden nur verarbeitet, solange die State Machine läuft. Die Events gehen jedoch nicht verloren, wenn die State Machine nicht läuft, sie werden in eine Queue gespeichert und beim Start der State Machine verarbeitet. Die State Machine kann jederzeit gestartet oder gestoppt werden.

Erstellen einer bbv.Common State Machine

State Machines können sehr einfach definiert werden, denn es sind nur wenige Schritte nötig.

Die einzelnen Schritte

Erstellen einer neuen Instanz der Aktiven oder Passiven State Machine.

Transitions definieren aus welchem State bei welchem Ereignis in welchen State gewechselt wird.

Aktionen legen fest, welche Funktion aufgerufen wird wenn ein bestimmter Status betreten oder verlassen wird.

Initialisierung legt den Start State fest.

Start startet die State Machine.

2.3.3 Alternativen im .Net Bereich

Simple State Machine ist eine in C# entwickelte Bibliothek. Die Elemente der State Machine werden in der Sprache Boo deklariert. Durch Simple State Machine wird definiert welche Zustände es gibt, welche Trigger diese Zustände kennen und in welchen Zustand beim auslösen dieses Triggers gewechselt wird. Für Simple State Machine existiert keine offizielle Dokumentation. Die Funktionalitäten sollen anhand der integrierten Unit Tests studiert werden. Es gibt auch ein einfaches Tutorial welches hilft die Grundfunktionalitäten zu verstehen. Simple State Machine wurde entwickelt als einfacher Ersatz für Workflow Foundation. Da Workflow Foundation für einfachste State Machine zu komplex ist und zu viel Overhead produziert.

Stateless stellt die wichtigsten Grundlagen zur Verfügung, um schnell und einfach State Machines realisieren zu können. Für jeden State kann definiert werden, was passiert wenn in diesen Zustand gewechselt wird, wenn der Zustand verlassen wird und welche Trigger der Zustand kennt. In jedem Zustand wird festgelegt bei welchem Trigger in welchen Zustand gewechselt wird. Durch die einfache Struktur bleibt die State Machine übersichtlich, jedoch beschränkt sich Stateless auf diese Grundfunktionen. Erweiterte Funktionalitäten müssen selber Implementiert werden. Stateless eignet sich für einfache State Machine welche später nicht verändert werden müssen. Stateless bietet keine Code Separation Möglichkeit. Alle Elemente müssen direkt im Code definiert werden.

2.3.4 Performancevergleich Windows WF und bbv.Common

Diese beiden im letzten Kapitel vorgestellten Laufumgebungen für State Machines, die Workflow Foundation von Microsoft und die bbv.Common Hierarchical State Machine, werden in diesem Kapitel durch Messungen verglichen.

Dafür wird für beide Systeme die identische State Machine erstellt. Durch das Auslösen von Events wird die Performance getestet. Da unbekannte Events sowohl bei Windows

WF wie auch bei bbv.Common verworfen werden, muss garantiert sein, dass immer ein Zustandswechsel vollzogen wird. Ansonsten würde der Event nicht dieselbe Verarbeitungszeit generieren, was die Messung verfälscht.

Test State Machine: Für den Test wird auf beiden Systemen eine State Machine betrieben, die 5 Zustände hat und von jedem Zustand in jeden anderen wechseln kann. Das folgende Bild zeigt die State Machine:

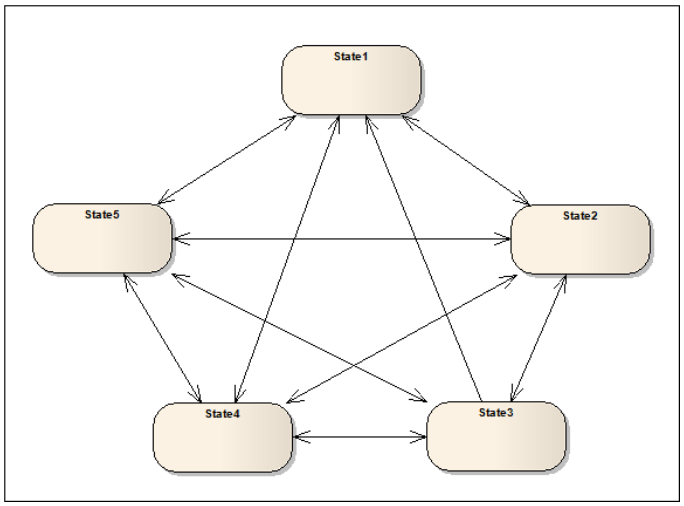


Abbildung (2.1) Zu testende State Machine

Vorgehen: Die oben beschriebene State Machine wird in beiden Umgebungen gleich implementiert, um die Verarbeitungszeit und den Speicherverbrauch zu messen. Dabei werden die Tests mit einer unterschiedlichen Anzahl ausgelöster Events durchgeführt, welche vorgefertigt in einer Liste an den EventManager übergeben werden. So kann die Verarbeitungszeit für die Bearbeitung eines Events mit entsprechenden Statusübergängen in der State Machine gemessen werden.

Die Verarbeitungszeit messen wir zum Einen über die gesamte Ausführungsperiode. Als zweiten Wert wollen wir auch die durchschnittliche Zeit für einen Statusübergang ermitteln.

Definieren der State Machine: Wie eine State Machine mit Windows WF modelliert wird ist in (siehe Seite 10) beschrieben. Bei 5 States ist die State Machine noch übersichtlich, wie in der folgenden Graphik zu sehen ist. Allerdings müssen in jedem State die möglichen Übergänge definiert werden, was in der Grundeinstellung einen grossen Aufwand zur Folge hat.

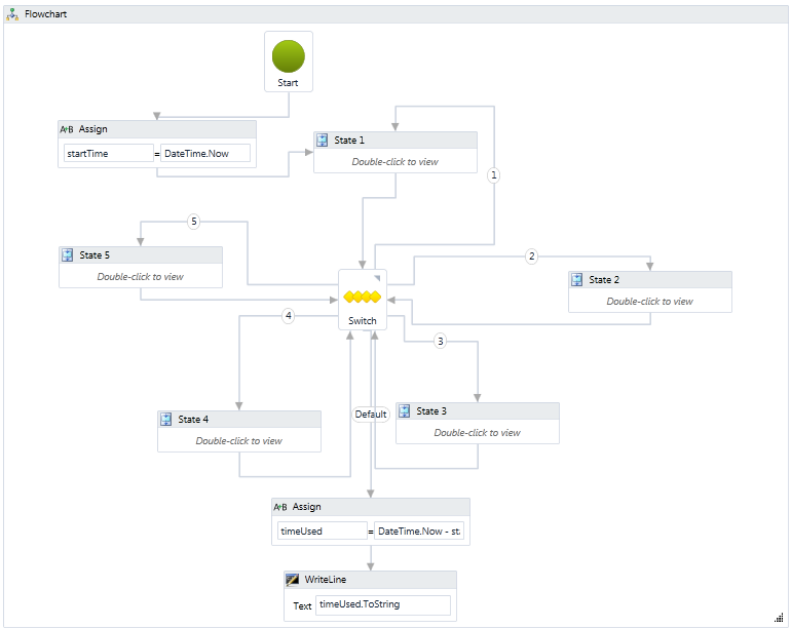


Abbildung (2.2) Gesamtsicht der State Machine

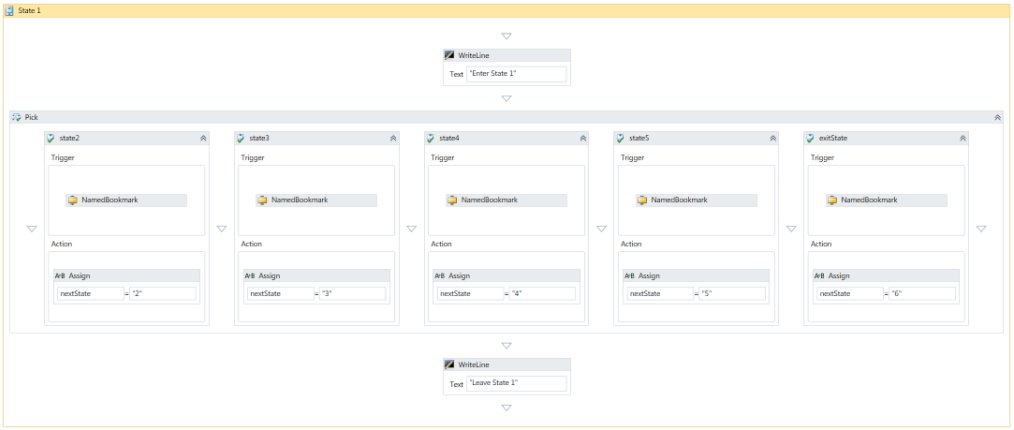


Abbildung (2.3) Übergangsdefinition im State

Das erstellen der bbv.Common State Machine ist demgegenüber wie in (siehe Seite 12) beschrieben einfacher automatisierbar, da die Übergänge mittels Funktionen definierbar sind. Dies lässt sich gut mit einer Schleife bewerkstelligen.

Messungen: Die folgenden Bilder zeigen das gemessene Laufzeitverhalten der *Test State Machine:* (siehe Seite 14) . Zu beachten ist der sehr grosse Unterschied der Anzahl getesteter Events. Bei der bbv.Common Messung sind es jeweils tausend Mal mehr Events.

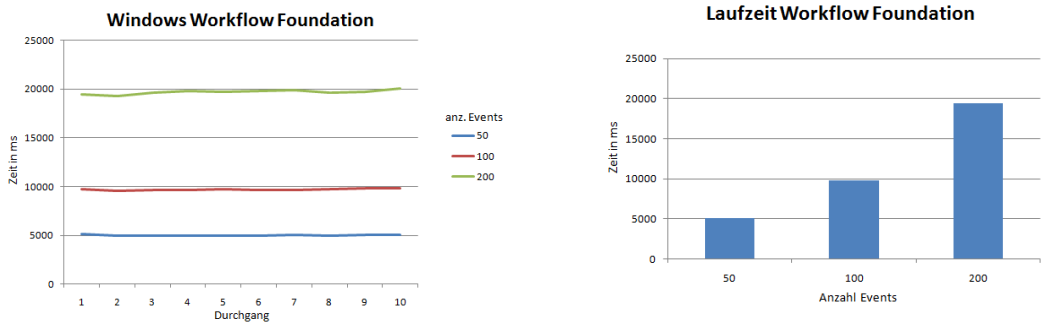


Abbildung (2.4) Laufzeitmessung mit WF

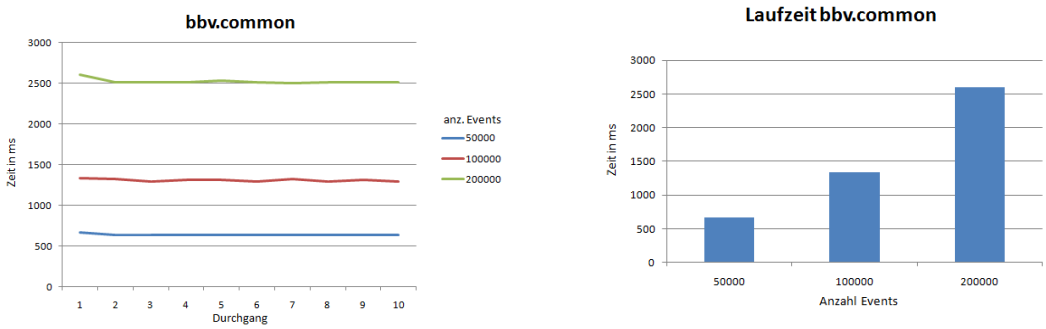


Abbildung (2.5) Laufzeitmessung mit bbv.Common

Daraus ergeben sich folgende Verarbeitungszeiten für einen einzelnen Event:

WF: 98ms / Event

bbv: 0.014ms / Event

Auswertung: Gut zu sehen ist, dass sowohl bei WF wie auch bei bbv.Common die Verarbeitungszeit pro Event konstant bleibt. Somit Skaliert das System linear zu Anzahl auszuführender Events. In diesem Punkt sind beide gleichwertig.

Allerdings ist die Implementation der State Machine in bbv.Common sehr viel schneller als im WF. Wie in obigen Messungen zu sehen ist, wurden mit der Standardimplementation für State Machines in WF pro Sekunde ungefähr 10 Events bearbeitet. Demgegenüber konnte die Implementation mit bbv.Common pro Sekunde ungefähr 70'000 Events verarbeiten.

Wichtig an diesen Messungen ist, dass in beiden Fällen keine Aktionen bei Betreten oder Verlassen der Zustände ausgeführt wurden. Ein Durchlauf mit bbv.Common, in dem

bei Betreten des neuen States eine Meldung auf der Konsole ausgegeben wird, hat die Ausführungszeit verfünffacht.

Diese Messungen zeigen, wie effizient die Implementation der bbv.Common State Machine arbeitet und dass sie somit in jedem Fall der Version von Windows WF vorzuziehen ist. Zusätzlich wird diese Entscheidung durch die sehr viel einfachere Erstellung der State Machine in der bbv.Common Bibliothek, welche weiter oben beschrieben ist.

2.4 Fazit

Wie im Kapitel Problemstellung beschrieben, bildet die Presence Engine mit den zu verwaltenden State Machines den Kern unserer Applikation und ist deshalb das wichtigste Kriterium für die Entscheidung zu Gunsten einer Technologie. In die engere Auswahl kamen von den oben aufgeführten Frameworks nur Activiti und Microsoft .Net mit der Workflow Foundation (WF) und dem bbv.Common Framework für State Machines. Die anderen Kandidaten haben eindeutige Argumente gegen eine Verwendung in diesem Projekt wie zum Beispiel die Lizenz bei den Open Source Frameworks.

Das einzige oben beschriebene Framework, welches wirklich für State Machines ausgelegt ist, ist dasjenige von bbv.Common. Dies zeigt sich sowohl beim erstellen, welches einfacher ausfällt, als auch beim Betrieb der State Machines. Wie beim *Performancevergleich Windows WF und bbv.Common* (siehe Seite 13) zu sehen ist diejenige von bbv sehr viel performanter als WF. Dies ist darauf zurückzuführen, dass WF eher auf sequentielle Workflows ausgelegt ist und weniger auf eventbasierte Zustandsautomaten. Dasselbe trifft auch auf activiti zu, welches wie WF für Workflows ausgelegt ist.

Die anderen Aspekte unserer Applikation unterstützen die Wahl von Microsoft .Net als Technologie. Im Bereich der deskriptiven Spezifikationssprache bietet XAML genauso wie die anderen erwähnten Sprachen alles um State Machines zu beschreiben und ist somit genauso geeignet. Allerdings bietet der Workflow Designer, der wie in (siehe Seite 11) beschrieben in eigene WPF basierte Applikationen eingebunden werden kann, eine Möglichkeit, die von den anderen Produkten nicht geboten wird. Was die Modularität und Erweiterbarkeit der Anbindung anderer Systeme betrifft gibt es für Java OSGi und von Microsoft MEF, von denen beide die von uns gebrauchten Funktionalitäten ermöglichen. Und für die Schnittstellenbeschreibung bietet Microsoft mit WCF auch alles von uns Benötigte.

Somit fällt die Wahl der Technologie auf Microsoft .Net mit einer auf der bbv.Common Hierarchical State Machine basierten Presence Engine.

3 Architektur Analyse

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	04.04.2011	Ricardo Alvarez	Dokument erstellt
0.2	07.04.2011	Ricardo Alvarez	Architekturbeschreibungen
0.3	11.04.2011	Ricardo Alvarez	Überarbeitung gemäss Review
0.4	12.04.2011	Ricardo Alvarez	Test Recheneffizienz
0.5	14.04.2011	Reto Brühwiler	Test Speicherbedarf
0.6	17.04.2011	Ricardo Alvarez	Review und Testergebnis

3.1 Einleitung

Die Presence Engine muss den Status von sehr vielen Objekten verwalten. Dies können bis zu mehreren hunderttausend Objekte sein. Dies bedingt eine möglichst Effiziente Architektur der verschiedenen Komponenten und deren Kommunikation miteinander. Dabei sind die Hauptkriterien zum einen die Verarbeitungseffizienz, zum anderen die Speichereffizienz.

In diesem Kapitel werden verschiedene Architekturen auf Tauglichkeit für die Presence Engine analysiert. Dafür wird jeweils die Geschwindigkeit der Verarbeitung wie auch der Speicherbedarf gemessen und auf Skalierbarkeit geprüft.

3.2 Presence Engine

Die Presence Engine ist für Verwaltung und Betrieb der State Machines verantwortlich. Architektonisch bestehen hier zwei Ansätze:

1. Es wird für jedes zu überwachende Objekt eine eigene State Machine erstellt. Der Vorteil hierbei ist eine schnellere Verarbeitung der Zustandsänderungen. Der Nachteil ist ein viel höherer Speicherverbrauch.

2. Es wird für eine Objektgruppe eine State Machine erstellt. Voraussetzung ist, dass alle Objekte der Gruppe dieselben Zustände und Übergänge haben. Diese Variante braucht weniger Speicher als die erste, allerdings werden mehr Operationen bei Statusübergängen gebraucht. Dies hat einen grösseren Rechenaufwand zur Folge.

Nachfolgend werden diese beiden Ansätze untersucht. Als erstes werden die Architekturen jeweils mit Klassenbild und Sequenzdiagramm vorgestellt. Danach werden die Architekturen auf Kriterien wie Rechenaufwand oder Speicherbedarf mittels geeigneter Messmethoden untersucht und dokumentiert.

3.2.1 Eine State Machine pro Objekt

Bei dieser Variante hat jedes zu überwachende Objekt eine eigene State Machine. Wie im Klassendiagramm unten ersichtlich gibt es folgende Objekte:

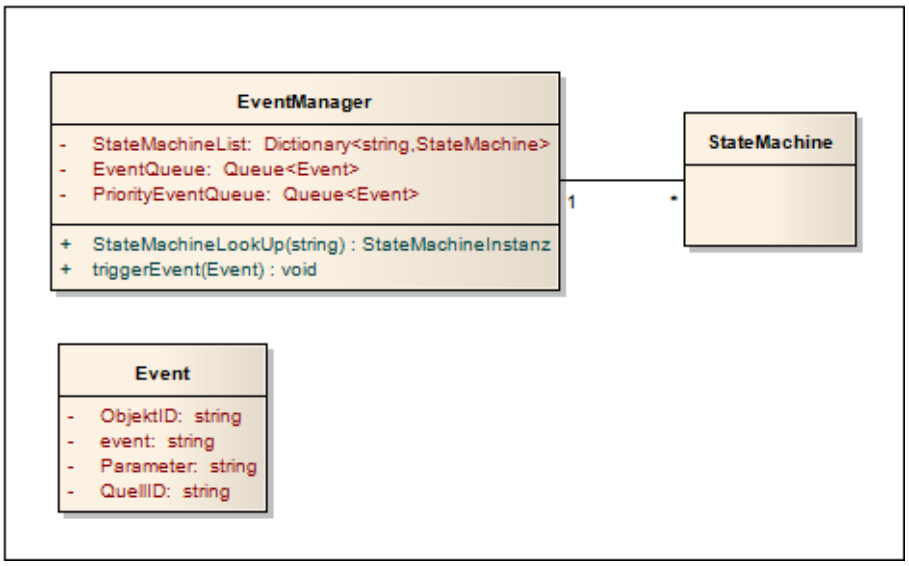


Abbildung (3.1) Klassendiagramm der Variante State Machine pro Objekt

EventManager: Der EventManager nimmt die Events der Fremdsysteme entgegen. Falls in einer gewissen Zeitspanne mehr Events kommen als verarbeitet werden können, werden sie in einer Queue zwischengespeichert. Priorisierte Events werden dabei in einer speziellen Queue gespeichert, welche zuerst verarbeitet wird. So können dringende Events möglichst schnell bearbeitet werden.

Für die Weiterleitung der Events sucht der Manager in der StateMachineList aufgrund der ZielID nach der korrekten State Machine und leitet die Nachricht weiter.

StateMachine: Die StateMachine verarbeitet den Event. Das heisst, sie macht den entsprechenden Zustandsübergang und löst die Funktionen aus, welche definiert sind.

Event: Der Event enthält alle benötigten Informationen, um dem EventManager die Weiterleitung und der State Machine die Verarbeitung zu ermöglichen.

Verarbeitungsablauf

Das folgende Sequenzdiagramm visualisiert diesen Ablauf mit den entsprechenden Aufrufen auf den Objekten.

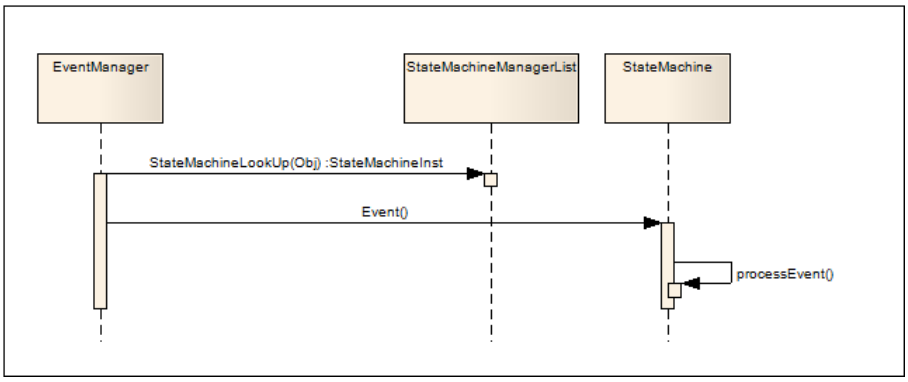


Abbildung (3.2) Sequenzdiagramm der Variante State Machine pro Objekt

Die erste Funktion erlaubt dem EventManager die Referenz der korrekten State Machine zu erhalten. Danach leitet er den Event weiter. Die Verarbeitung des Events geschieht dann intern in der State Machine, wo auch die ausgehenden Aktionen ausgelöst werden.

3.2.2 Eine State Machine pro Objektgruppe

Bei dieser Variante werden die Objekte mit demselben Ablauf zu einer Gruppe zusammengefasst, die dann mit nur einer State Machine verwaltet werden. Das folgende Klassendiagramm zeigt den Aufbau dieser Architektur.

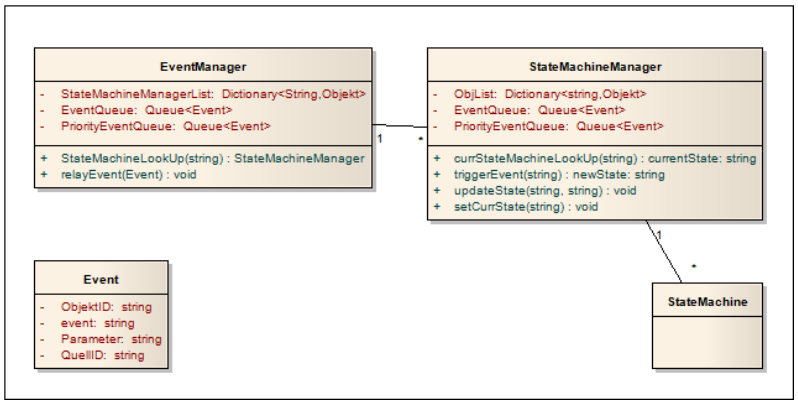


Abbildung (3.3) Klassendiagramm der Variante eine State Machine pro Objektgruppe

EventManager: Der EventManager hat in dieser Architektur dieselbe Aufgabe wie bei 3.2.1 *EventManager:* (auf der vorherigen Seite) beschrieben, nämlich Events der Fremdsysteme entgegenzunehmen und zu verteilen.

Allerdings gibt er sie nicht direkt an die State Machine weiter, sondern an den StateMachineManager, der die Objektgruppe verwaltet.

StateMachineManager: Der StateMachineManager übernimmt den Event vom EventManager. Mit Hilfe der ZielID ermittelt er in der Liste den momentanen Status des Zielobjekts. Damit versetzt er die State Machine in den benötigten Startzustand, also den vorher ermittelten momentanen Zustand des Objekts.

Jetzt kann er den Event an die State Machine weiterleiten, welche für die Verarbeitung zuständig ist.

StateMachine: Die StateMachine verarbeitet den Event. Das heisst, sie macht den entsprechenden Zustandsübergang und löst die Funktionen aus, welche definiert sind. Es hat gegenüber der Variante *Eine State Machine pro Objekt* (siehe Seite 20) keine Änderung gegeben.

Event: Der Event enthält alle benötigten Informationen, um dem EventManager die Weiterleitung und der State Machine die Verarbeitung zu ermöglichen. Es hat gegenüber der Variante *Eine State Machine pro Objekt* (siehe Seite 20) keine Änderung gegeben.

Verarbeitungsablauf

Das folgende Sequenzdiagramm visualisiert oben beschriebenen Ablauf mit den entsprechenden Aufrufen auf den Objekten.

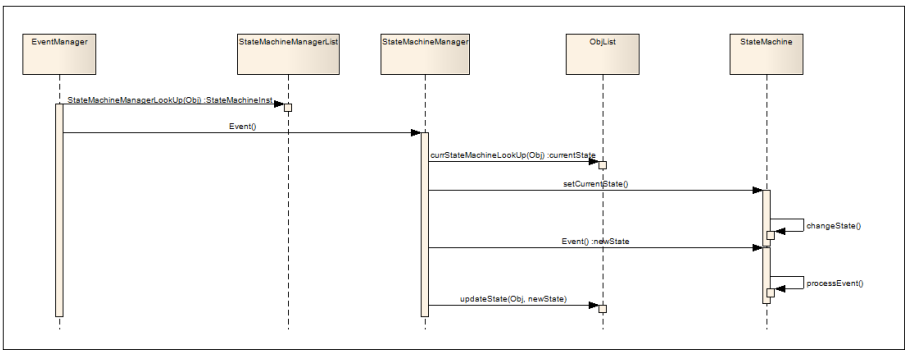


Abbildung (3.4) Sequenzdiagramm der Variante eine State Machine pro Objektgruppe

Die erste Funktion erlaubt dem EventManager die Referenz des korrekten StateMachineManagers zu erhalten und den Event an ihn weiterzuleiten. Dieser ermittelt den momentanen Zustand des Objekts und versetzt die State Machine in diesen, damit der Event verarbeitet werden kann. Dies geschieht wie bei *Eine State Machine pro Objekt* (siehe Seite 20) intern in der State Machine, wo auch die ausgehenden Aktionen ausgelöst werden.

Probleme dieser Architektur

Beim Umsetzen der gerade beschriebenen Architektur für die Variante eine State Machine pro Objektgruppe ist folgendes Problem entstanden:

Mit dem ersten Aufruf an die State Machine wird sie in den momentanen Zustand des Objektes versetzt. Bei diesem allfälligen Zustandsübergang darf aber keine Aktion ausgelöst werden, da er nur als Vorbereitung für den eigentlichen Event dient. Um dies zu erreichen müssten Hilfszustände eingerichtet werden, die diese Übergänge ermöglichen.

Durch diesen Mechanismus wird aber ermöglicht, dass ein Objekt wieder in den Zustand eintritt, in dem er vorher war. Dabei werden die Aktionen dieses Zustandes erneut ausgeführt, obwohl dies gar nicht nötig ist. Da der StateMachineManager aber nicht weiss, in welchen Zustand der auszuführende Event führt, kann dies nicht unterbunden werden.

Lösungsansätze: Der erste Lösungsansatz war, die State Machine intern so anzupassen, dass der momentane Zustand des Objekts direkt eingestellt werden kann und danach der Event Verarbeitet wird. Dieser Ansatz wäre von der Verarbeitungszeit sehr effizient, da nur ein Aufruf an die State Machine nötig wäre, also ein Verarbeitungszyklus eingespart werden könnte. Allerdings müsste die bbv Bibliothek geändert werden, um einen direkten Zugriff auf den momentanen Status zu erlauben. Diese Anpassung der Bibliothek wäre grundsätzlich möglich, hat aber auch Nachteile. So könnte bei einer neuen Version der bbv.Common Bibliothek diese erst übernommen werden, wenn dieselben Anpassungen wieder vorgenommen worden sind. Deshalb haben wir uns entschieden, eine andere Lösung anzustreben, was zur im Folgenden beschriebenen alternativen Architektur.

3.2.3 Alternative Architektur einer State Machine pro Objektgruppe

Diese Variante funktioniert gleich wie 3.2.2 *Eine State Machine pro Objektgruppe* (siehe Seite 21) mit dem Unterschied, dass die State Machine nur den Zustand des Objekts verwaltet und an den StateMachineManager zurück gibt. Dieser ist dann für das Auslösen der Aktionen zuständig.

Das folgende Klassendiagramm veranschaulicht die Komponenten:

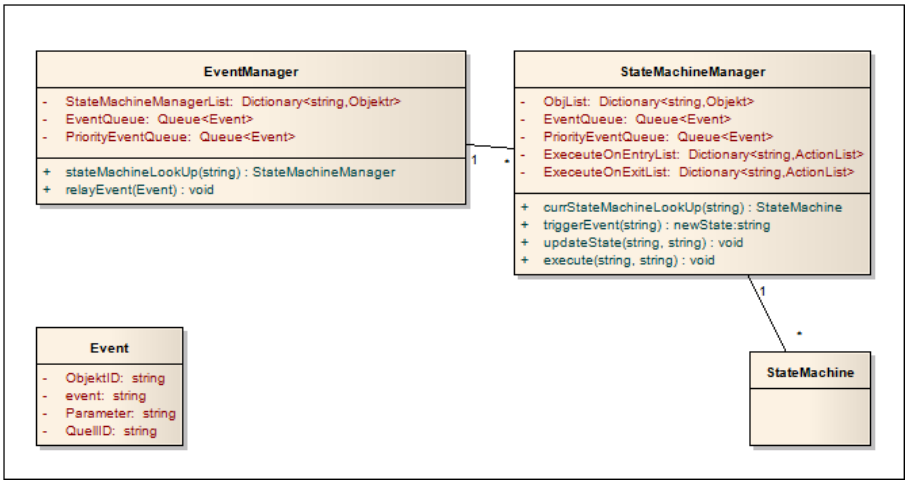


Abbildung (3.5) Klassendiagramm der zweiten Variante State Machine pro Objektgruppe

EventManager: Der EventManager hat in dieser Architektur dieselbe Aufgabe wie bei 3.2.2 *EventManager:* (siehe Seite 21) beschrieben, nämlich Events der Fremdsysteme entgegenzunehmen und an die StateMachineManager zu verteilen.

StateMachineManager: Der StateMachineManager übernimmt den Event vom EventManager. Mit Hilfe der ZielID ermittelt er in der Liste den momentanen Status des Zielobjekts. Damit versetzt er die State Machine in den benötigten Startzustand, also den vorher ermittelten momentanen Zustand des Objekts.

Jetzt kann er den Event an die State Machine weiterleiten, welche den Event verarbeitet und den neuen Zustand des Objekts zurückgibt. Jetzt löst der StateMachineManager die benötigten Aktionen für das Objekt aus.

StateMachine: Die StateMachine verarbeitet den Event. Das heisst, sie macht den entsprechenden Zustandsübergang. Allerdings löst sie in dieser Variante keine Aktionen aus, sondern informiert nur den StateMachineManager über den neuen Zustand des Objekts.

Event: Der Event enthält alle benötigten Informationen, um dem EventManager die Weiterleitung und der State Machine die Verarbeitung zu ermöglichen. Es hat gegenüber der Variante *Eine State Machine pro Objekt* (siehe Seite 20) keine Änderung gegeben.

Verarbeitungsablauf

Das folgende Sequenzdiagramm visualisiert oben beschriebenen Ablauf mit den entsprechenden Aufrufen auf den Objekten.

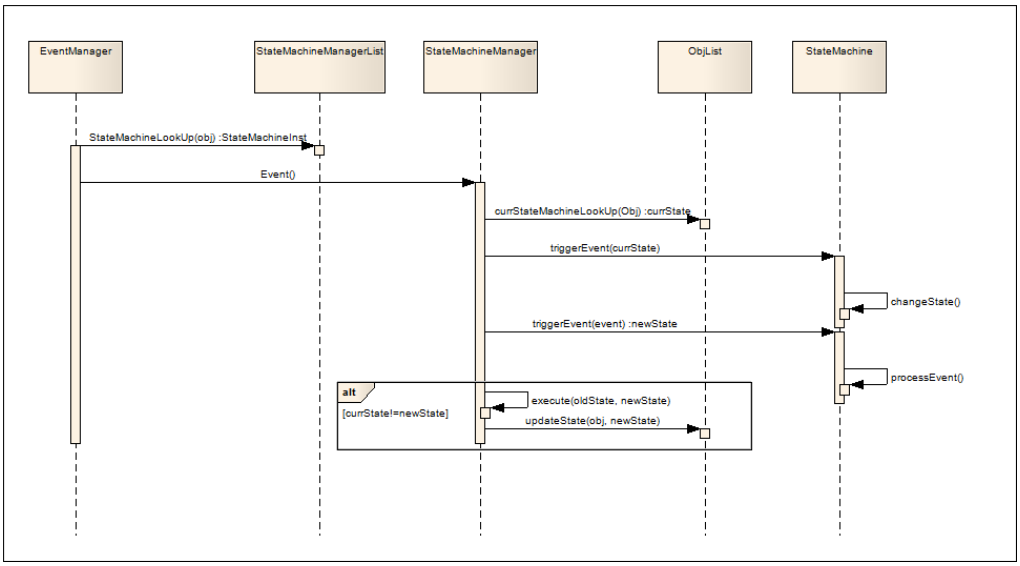


Abbildung (3.6) Sequenzdiagramm der zweiten Variante State Machine pro Objektgruppe

Die erste Funktion erlaubt dem EventManager die Referenz des korrekten StateMachineManagers zu erhalten und den Event an ihn weiterzuleiten. Dieser ermittelt den momentanen Zustand des Objekts und versetzt die State Machine in diesen Zustand. Jetzt kann er den richtigen Event weiterleiten, damit die State Machine diesen verarbeiten und den neuen Zustand zurückgeben kann. Falls das Objekt in einen anderen Zustand gewechselt ist, kann jetzt der StateMachineManager die definierten Aktionen auslösen. Dies sind die Aktionen bei Austritt aus dem alten Zustand und diejenigen bei Eintritt in den neuen Zustand.

3.2.4 Test der Architekturen

Um die in 3.2 *Presence Engine* (siehe Seite 19) beschriebenen Architekturformen der Presence Engine zu testen wird ein Prototyp erstellt. In diesem werden die beiden oben vorgestellten Architekturen *Eine State Machine pro Objekt* (siehe Seite 20) und *Alternative Architektur einer State Machine pro Objektgruppe* (siehe Seite 24) realisiert.

Der Prototyp ist so gestalten, dass folgende Parameter des Testszenarios angepasst werden kann:

- Die Anzahl States der State Machine.
- Die Anzahl der zu überwachenden Objekte
- Die Anzahl State Machines.
- Die Anzahl Events und der Warteschlange.
- Die Anzahl Durchläufe desselben Tests.

Diese Parameter gelten dann für beide Implementationen, so dass ein Vergleich möglich wird.

Performancetest

Bei diesem Test wird gemessen, wie lange die Engine für die Verarbeitung der Events benötigt. In verschiedenen Durchläufen wird die Auswirkung der Anzahl Events, States und State Machines gemessen, so dass auch Aussagen über die Skalierbarkeit des Systems gemacht werden können.

Übersicht der Tests:

1. *Einfluss der Grösse der State Machine:* (auf dieser Seite)
2. *Eventverarbeitung ohne Aktionsauslösung:* (auf der nächsten Seite)
3. *Eventverarbeitung mit Aktionsauslösung:* (siehe Seite 29)
4. *Eventverarbeitung mit zufälligen Events:* (siehe Seite 30)

Einfluss der Grösse der State Machine: In diesem ersten Test wollen wir messen, ob die Grösse und Form der State Machine einen Einfluss auf die Performance hat. Zu diesem Zweck haben wir die Anzahl der States angepasst. Für die erste Messung haben wir 5 States verwendet, danach 10 und 15. Weil wir Übergänge von jedem State zu jedem anderen hinzufügen, vergrössert dies die State Machine jeweils um ein Vielfaches.

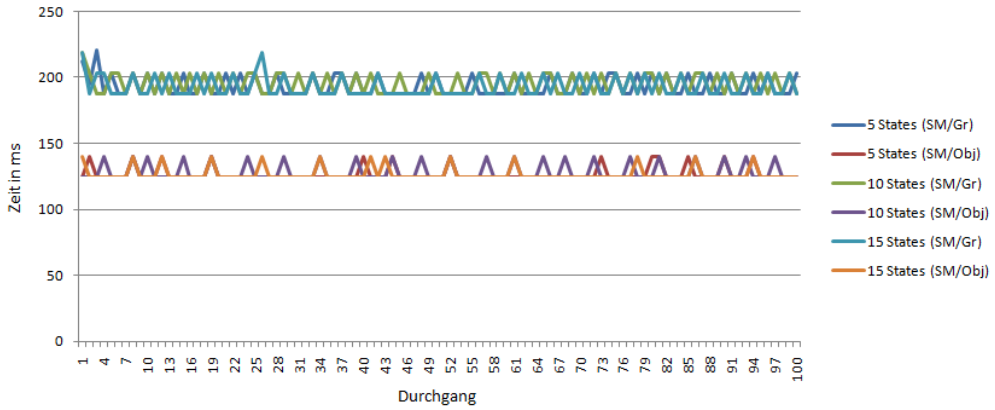


Abbildung (3.7) Einfluss der Anzahl States pro State Machine: Konstanz der Verarbeitungszeit

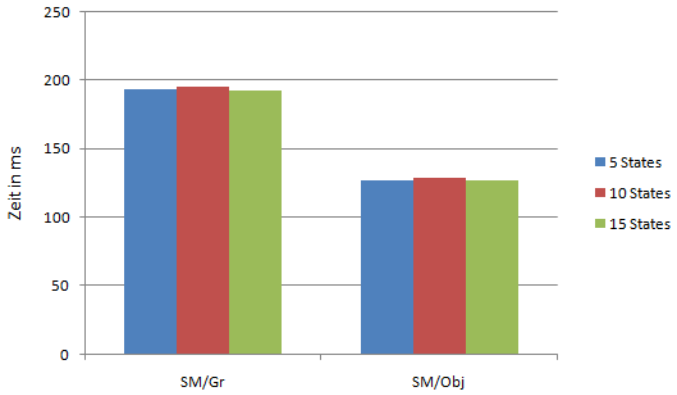


Abbildung (3.8) Einfluss der Anzahl States pro State Machine: Durchschnittliche Verarbeitungszeit

Wie die Graphik oben zeigt, hat die Anzahl der States keinen Einfluss auf die Verarbeitungsgeschwindigkeit. Dies gilt für beide Architekturformen.

Eventverarbeitung ohne Aktionsauslösung: Ziel dieser Messungen ist es, die Verarbeitungszeit der Events ohne Auslösen von Aktionen zu messen. Dies Entspricht der reinen Ausführungszeit für Statusübergänge und zeigt den Effizienzunterschied zwischen den beiden Implementationsformen. Die Variante der State Machine pro Objektgruppe löst zwei Statusübergange aus um einen Event zu verarbeiten, die Variante eine State Machine für ein Objekt löst nur eine Statusübergang aus, also halb so viele.

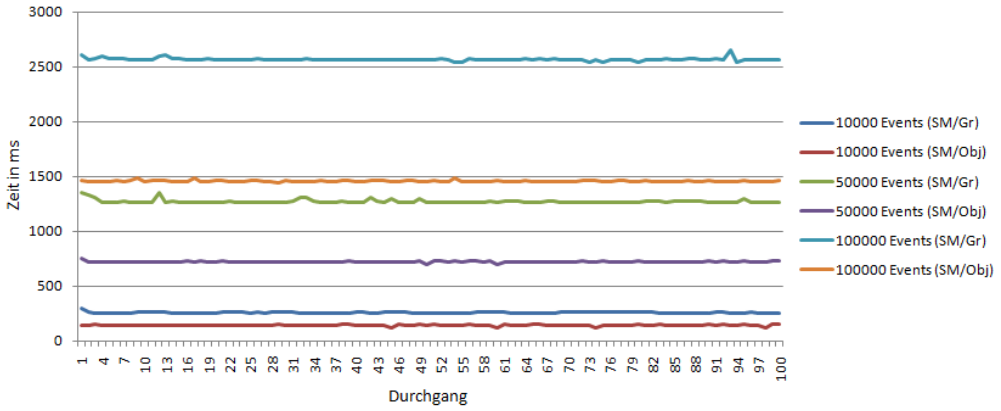


Abbildung (3.9) Vergleich der Verarbeitungszeit ohne ausgelöste Aktionen: Konstanz der Verarbeitungszeit

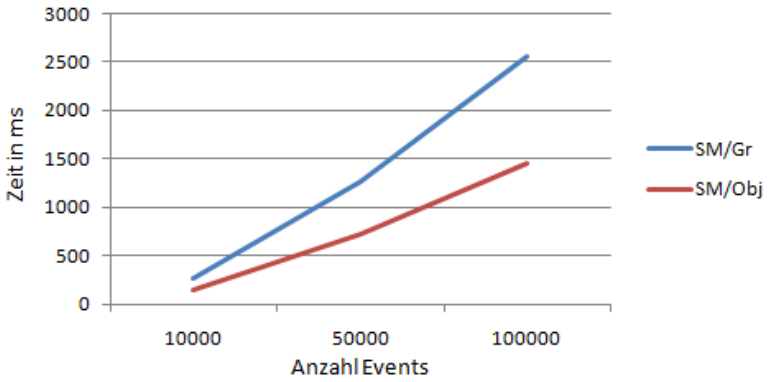


Abbildung (3.10) Vergleich der Verarbeitungszeit ohne ausgelöste Aktionen: Durchschnittliche Verarbeitungszeit

Wie in der Graphik ersichtlich ist, können von beiden Varianten in sehr kurzer Zeit sehr viele Events verarbeitet werden. Die State Machine ist also sehr effizient.

Der direkte Vergleich zeigt, dass die Gruppenvariante beinahe doppelt so lange für die Verarbeitung der Events benötigt als die andere Variante. Dies ist durch den oben erwähnten Umstand zu erklären, dass die Gruppenvariante für jeden Event zwei Verarbeitungszyklen braucht. Einen zum einstellen des Objektzustandes, der zweite für die Verarbeitung des eigentlichen Events.

Weiter ist in der Graphik zu sehen, dass beide Varianten linear mit der zu verarbeitenden Anzahl Events skalieren. Die Erhöhung der Anzahl Events um den Faktor 10 hat eine Erhöhung der Verarbeitungszeit um den Faktor 10 zur Folge. Dieses Verhalten ist sehr gut und gilt unabhängig von Grösse oder Form der State Machine wie oben gezeigt.

Die reine Eventverarbeitung und Zustandsänderung der State Machine läuft also schnell und effizient ab und hat eine hohe Skalierbarkeit. Als nächstes soll noch der Einfluss auszulösender Aktionen auf die Performance gemessen werden.

Eventverarbeitung mit Aktionsauslösung: Diese Messung hat das Ziel, den Einfluss von ausgelösten Aktionen auf die Verarbeitungszeit beurteilen zu können. Zu diesem Zweck werden den States Aktionen hinzugefügt. Zum Vergleich haben wir sowohl rechenintensive Aktionen wie das Schreiben in die Konsole ausprobiert als auch einfache Funktionen wie das Erstellen eines Events und Speichern in eine Collection.

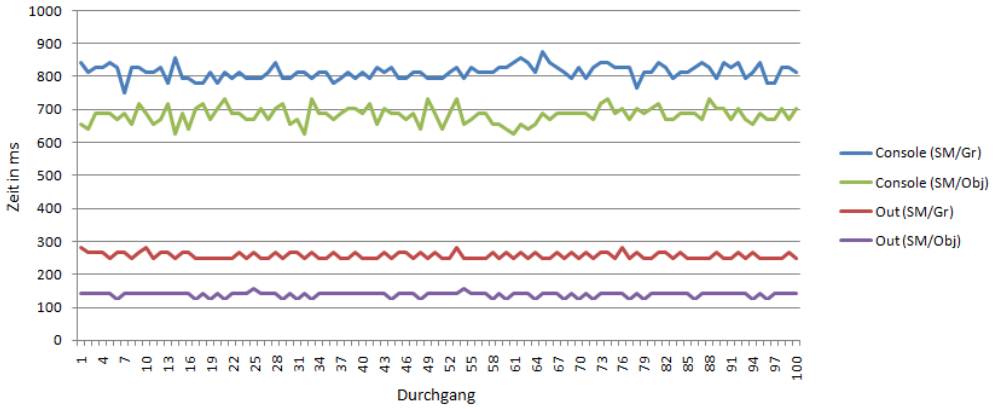


Abbildung (3.11) Vergleich der Verarbeitungszeit mit ausgelösten Aktionen: Konstanz der Verarbeitungszeit

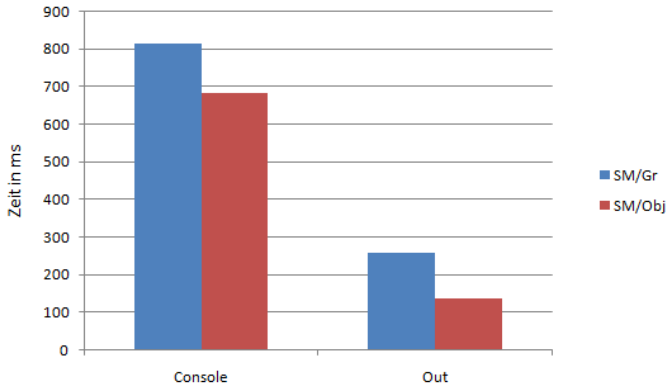


Abbildung (3.12) Vergleich der Verarbeitungszeit mit ausgelösten Aktionen: Durchschnittliche Verarbeitungszeit

Die Graphik zeigt, dass der Einfluss der ausgelösten Aktion ziemlich stark sein kann. Das Schreiben in die Konsole erhöht die Verarbeitungszeit circa um den Faktor 5 wohingegen

das erstellen eines neuen Events oder das Speichern in eine Collection zeitlich praktisch nicht zu erfassen sind.

Da diese Aussage für beide Architekturen gilt, hat dieses Kriterium keinen Einfluss auf die Wahl der Architektur. Beide werden gleich stark ausgebremst oder nicht, je nach Wahl. Somit müssen wir zwar schauen, dass möglichst effiziente Aktionen eingebunden werden um die Performance des Systems möglichst wenig zu beeinträchtigen, allerdings unabhängig von der Wahl der Architektur.

Eventverarbeitung mit zufälligen Events: Ziel dieser Messung ist es zu zeigen, wie sich das System in einer nicht präparierten Umgebung verhält. Das heisst, sowohl der Event wie auch das betroffene Objekt sind komplett zufällig gewählt.

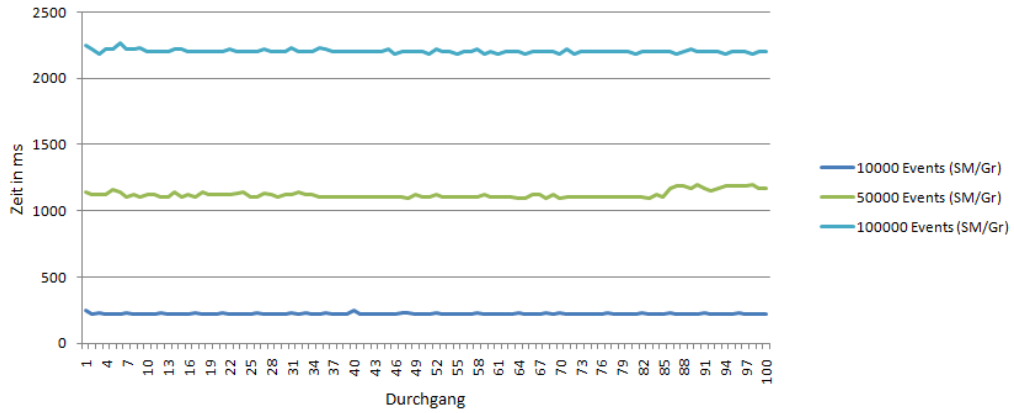


Abbildung (3.13) Vergleich der Verarbeitungszeit mit zufälligen Events: Konstanz der Verarbeitungszeit

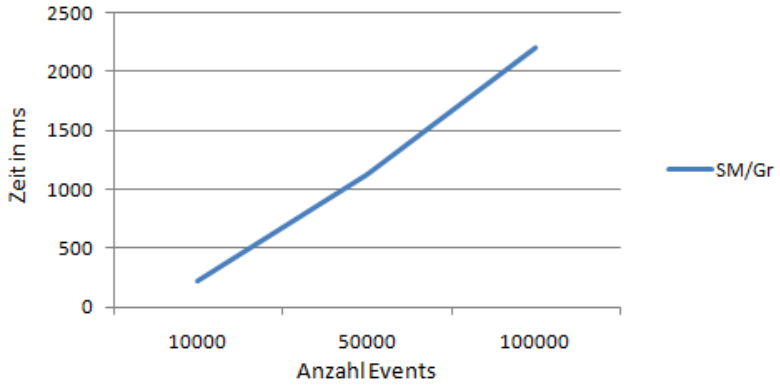


Abbildung (3.14) Vergleich der Verarbeitungszeit mit zufälligen Events: Durchschnittliche Verarbeitungszeit

Wie zu sehen ist variiert die Verarbeitungszeit wenig bei den verschiedenen Durchläufen. Dies ist dadurch zu erklären, dass bei zufälligen Event Listen vereinzelt der erste Event zum setzen des Startzustandes gespart wird, da die State Machine bereits im richtigen Zustand ist. Allerdings ist diese Einsparung dank der generell sehr schnellen Verarbeitung zu vernachlässigen im Verhältnis zur gesamten Verarbeitungszeit, sofern diese Fälle nicht die Mehrheit bilden.

Dieses Verhalten ist auch für den Betrieb mit wirklichen Events zu erwarten.

Test des Speicherbedarfs

Um den Speicherbedarf messen zu können wurde der Prototyp so umgebaut, dass die Testparameter als Inputparameter der Main Funktion übergeben werden können. Der Prototyp wurde mit den verschiedenen Parameter laufen gelassen und der Speicherbedarf mit Hilfe des Resource Monitor, welcher Bestandteil von Windows ist, ausgelesen. Gemessen wurde dabei der durch das System reservierte Speicher.

Alle Messungen des Speicherverbrauchs wurden ohne Eventverarbeitung durchgeführt, damit der reine Speicherbedarf der State Machines ersichtlich ist. Events hätten diese Angabe verfälscht, da sie auch Platz brauchen.

Übersicht der Tests:

1. *Einfluss der Anzahl States auf Speicherbedarf:* (auf dieser Seite)
2. *Einfluss der State Machines auf den Speicherbedarf:* (auf der nächsten Seite)
3. *Einfluss der Anzahl Objekte auf den Speicherbedarf:* (auf der nächsten Seite)

Einfluss der Anzahl States auf Speicherbedarf: Ziel dieser Messung ist es, den Einfluss der Anzahl States auf den Speicherbedarf aufzuzeigen. Es wurde nur ein Objekt und somit auch nur eine State Machine erstellt. Dadurch unterscheiden sich die beiden Architekturvarianten nicht voneinander, was nur eine Messung nötig macht.

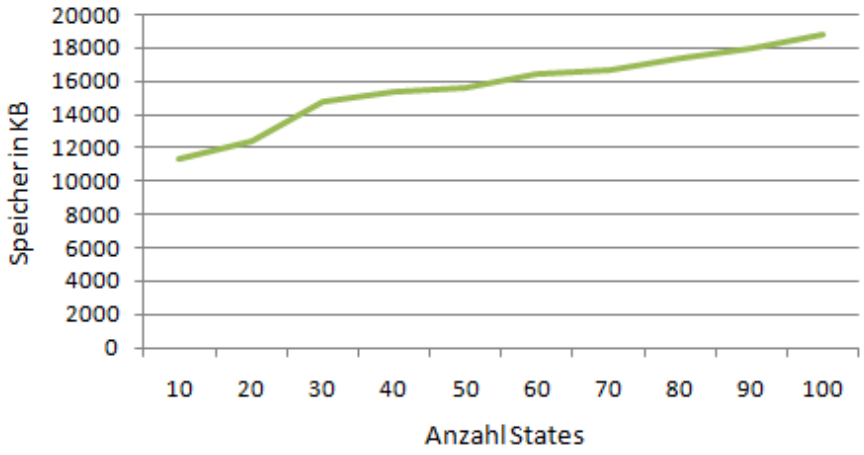


Abbildung (3.15) Einfluss der Anzahl States auf den Speicherbedarf

Wie auf der Grafik zu sehen ist steigt der Speicherbedarf beinahe linear mit der Anzahl States an.

Einfluss der State Machines auf den Speicherbedarf: Bei dieser Messung wurde der Speicherbedarf in Abhängigkeit zur Anzahl State Machines analysiert. Jede State Machine hatte 5 States und Übergänge von jedem State zu jedem anderen. Damit die Ergebnisse verglichen werden können wurden gleich viele State Machines wie Objekte erzeugt.

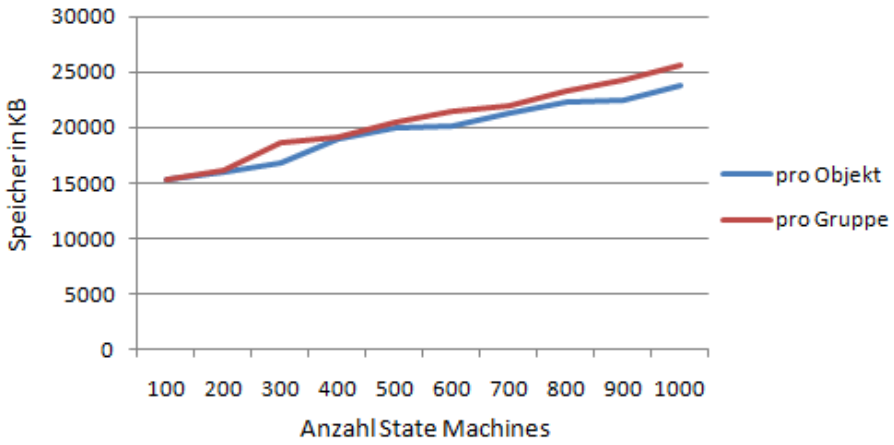


Abbildung (3.16) Einfluss der Anzahl State Machines auf den Speicherbedarf

Wie erwartet verhalten sich beide getesteten Varianten gleich. Bei beiden steigt der Speicherbedarf linear zur Anzahl State Machines an.

Einfluss der Anzahl Objekte auf den Speicherbedarf: Durch diesen Versuch wird analysiert, in welchem Verhältnis die Anzahl State Machines zur Anzahl Objekte stehen müssen, damit sich die Variante eine State Machine pro Objekt Gruppe lohnt. Getestet wurde mit 1000 zu überwachende Objekte und 5 States je State Machine.

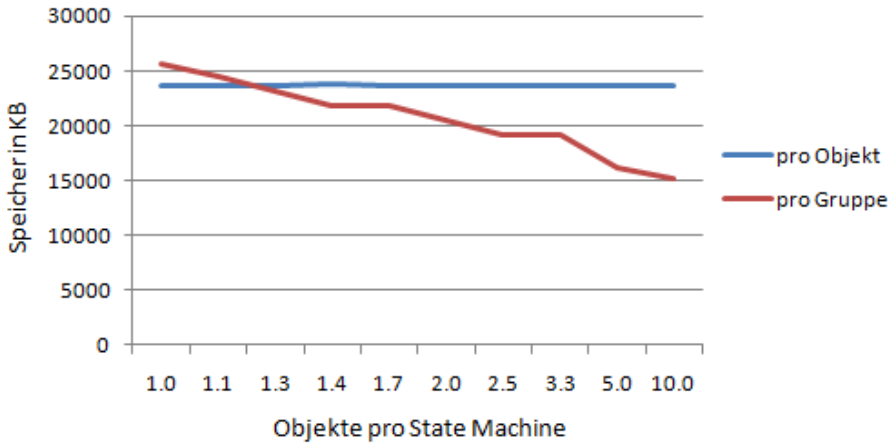


Abbildung (3.17) Einfluss des Anzahl Objekte pro State Machine Verhältnis auf den Speicherbedarf

Der Overhead der pro Gruppe Variante wird sehr schnell durch die geringeren Anzahl State Machines aufgehoben. Somit lohnt sich diese Variante bereits ab 2 Objekten pro State Machine, wie auf der Grafik ersichtlich ist.

3.2.5 Ergebnisauswertung

Die Messungen der Performance haben folgende Erkenntnisse gebracht:

- Die Verarbeitungszeiten sind über jeweils 100 Testläufe sehr konstant geblieben.
- Die durchschnittlichen Verarbeitungszeiten der Variante mit einer State Machine pro Objektgruppe sind wegen der zwei Eventverarbeitungen annähernd doppelt so lang.
- Werden Aktionen ausgelöst ist darauf zu achten, dass diese möglichst performant sind. Ansonsten wird das System stark gebremst wie am Beispiel mit der Konsolenausgabe ersichtlich ist.
- Die Verarbeitungszeit skaliert sehr gut mit der Anzahl Objekte. Das Verhältnis bleibt annähernd linear.

Die Messungen des Speicherbedarfs haben folgendes ergeben:

- Der Speicherbedarf verhält sich linear zur Grösse der State Machine.

- Ab 2 Objekten pro Gruppe lohnt sich speichertechnisch die Variante mit einer State Machine pro Objektgruppe.

Die Performance der Variante mit einer State Machine pro Objekt ist zwar schneller, aber die andere Variante ist auch noch sehr schnell und kann sehr viele Events in kurzer Zeit verarbeiten. Da fällt die Einsparung beim Speicher stärker ins Gewicht, da sehr viele Objekte mit dieser Presence Engine verwaltet werden sollen. Da im Schnitt mehr als 2 Objekte pro Objektgruppe zu erwarten sind eignet sich die Variante eine State Machine pro Gruppe.

Teil II

Umsetzung

4 Anforderungsspezifikation

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	23.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	23.03.2011	Reto Brüwihler	Nicht Funktionale Anforderungen
0.3	24.03.2011	Ricardo Alvarez	Funktionale Anforderungen
0.4	28.03.2011	Ricardo Alvarez	Use Cases und Überarbeitung
0.5	28.03.2011	Ricardo Alvarez	Überarbeitung gemäss Review
0.6	05.04.2011	Ricardo Alvarez	Überarbeitung gemäss Review

4.1 Allgemeine Beschreibung

4.1.1 Ziel und Zweck

Bisher wurde für jeden Anwendungsfall eine spezifische Presence Engine mit festen State Machines entwickelt, welche nur für diesen Kunden verwendet werden konnte. Somit mussten grosse Teile der Applikation für jedes Einsatzgebiet neu entwickelt werden, was einen grossen Initialaufwand zu Folge hatte.

Ziel dieses Projektes ist es, eine flexible Presence Engine zu entwickeln. State Machines sollen einfach beschrieben und betrieben werden können. Ausserdem soll sie so flexibel sein, dass weitere Fremdsysteme angebunden und so die Presence Engine erweitert werden kann oder bestehende Systeme geändert werden können.

4.1.2 Produkt Perspektive

Die Presence Engine soll flexibel genug gestaltet sein und dementsprechend Modifikationsmöglichkeiten bieten, dass sie mit annehmbarem Aufwand an ein neues Umfeld angepasst werden kann.

4.1.3 Produkt Funktion

Die State Machines sollen deklarativ beschrieben werden können. Dies soll durch ein Spezifikationsfile geschehen, welches von Hand oder mittels graphischen Editors erstellt wird. Ermöglicht wird dies durch eine entsprechend lesbare Spezifikationssprache. Statusänderungen der Objekte können von Fremdsystemen geliefert werden und bei Übergängen von einem State zu einem anderen können Aktionen ausgelöst werden. Um dies dynamisch zu gestalten soll die eingehende Schnittstelle offen genug definiert sein, um Inputs von

möglichst vielen verschiedenen Sensorsystemen verarbeiten zu können und damit neue Systeme eingebunden und Änderungen ausgelöst werden können. Die ausgehenden Aktionen sollen auch dynamisch erweitert werden können.

4.1.4 Benutzer Charakteristiken

Entwickler

Nutzungshäufigkeit	30 %
Nutzungsintensität	100 %
Wann	Bei Änderung des Einsatzgebietes
Erfahrung	Experte

Tabelle (4.1) Charakteristik Entwickler

Der Entwickler passt die Presence Engine an die Bedingungen des Einsatzgebietes an und modifiziert sie, falls sich das Umfeld ändert. Dies beinhaltet insbesondere die Erweiterungsmodule, welche die Funktionen der Applikation zur Kommunikation mit Umsystemen benötigt.

Verwalter

Nutzungshäufigkeit	70 %
Nutzungsintensität	30 %
Wann	Bei Anpassungen beim Kunden
Erfahrung	Versiert

Tabelle (4.2) Charakteristik Verwalter

Der Verwalter wird durch den Kunden gestellt und ist zuständig für kleinere Anpassungen an State Machines oder das Erstellen neuer State Machines, sofern keine neuen Komponenten oder Aktionen involviert sind. Dies bedingt detaillierte Kenntnisse der Abläufe und möglichen Aktionen. Eine weitere Aufgabe ist die Überwachung der Presence Engine.

4.2 Funktionale Anforderungen

4.2.1 Übersicht

Anforderung	Priorität	Status
Presence Engine	1	-
Neue State Machine instanziieren	1	-
Laufende State Machine ändern	1	-
State Machine deklarativ beschreiben	1	-
Deklarationen der State Machines verwalten	2	-
Schnittstellen definieren	1	-
Modulare Schnittstellenerweiterung ermöglichen	2	-
GUI für die Verwaltung erstellen	3	-

4.2.2 Deklaration der State Machine

Die State Machines sollen deklarativ beschrieben werden können. Diese Beschreibung soll möglichst einfach durch Menschen lesbar sein, so dass der Entwickler die benötigten State Machines direkt in dieser Sprache beschreiben kann. Die resultierende Beschreibung soll gespeichert und verändert werden können.

Desweiteren soll es möglich sein, neue Deklarationen von State Machines aus bestehenden zu generieren. Dafür soll eine bestehende Deklaration in den Editor geladen werden und als Basis dienen. Bereits bestehende, deklarierte State Machines sollen als Sub State Machines in andere State Machines eingebunden werden können. Dies garantiert eine Wiederverwendung und erleichtert die Entwicklung von State Machines mit gleichen Teilen.

Spezifikationssprache

Die Spezifikationssprache dient der deklarativen Beschreibung der State Machines. Sie soll wie oben beschrieben auch für Menschen möglichst einfach lesbar oder durch einen Editor darstellbar und konfigurierbar sein.

4.2.3 Presence Engine

Die Presence Engine bildet den Kern der Applikation. Sie soll die Deklarationen der State Machines einlesen und daraus die State Machines generieren. Diese verschiedenen State Machines werden dann durch die Engine betrieben und verwaltet. Insbesondere ist sie dafür zuständig, dass die Events zur richtigen State Machine geleitet werden. Kommt ein Event mit höherer Priorität, soll dieser auch bevorzugt behandelt und möglichst schnell weitergeleitet werden. Bei Statusübergängen sollen die benötigten Aktionen oder Nachrichten ausgelöst werden.

Instanziieren der State Machines: Durch das einlesen eines Deklarationsfiles wird eine Vorlage für ein State Machine Typ mit allen Zuständen und Übergängen erstellt. Diese Vorlage kann dann für ein spezifisches Objekt benutzt werden und als konkrete State Machine instanziiert werden. Für das erstellen dieser konkreten State Machines soll es zwei Möglichkeiten geben. Zum einen soll es manuell einstellbar sein, in dem die Vorlage gewählt, das Objekt mit der eindeutigen ID benennt und dann die State Machine gestartet wird. Dieser Vorgang soll aber auch automatisierbar sein. Mittels Event kann eine solche Instanziierung ausgelöst werden wenn im Fremdsystem ein neues Objekt erstellt wird, welches durch eine State Machine überwacht werden soll. Genauso wie die automatisch erstellten State Machines mittels Event instanziiert werden, sollen sie auch wieder automatisch mittels Event deaktiviert werden, wenn das Objekt nicht mehr aktiv ist.

Ändern einer laufender State Machine: Desweiteren soll es möglich sein Änderungen an bestehenden State Machines vorzunehmen. Ist eine solche State Machine gerade aktiv, soll der laufende Betrieb möglichst wenig gestört werden und eine konsistente Zustandsverwaltung garantiert sein. Es soll also möglich sein nur die betroffene State Machine zu unterbrechen und mit den Änderungen neu zu starten. Während der Inaktivitätsphase sollen Events für diese State Machine chronologisch korrekt zwischengespeichert werden, so dass sie nach dem Neustart in den korrekten Status wechseln kann. Dies muss dem System natürlich entsprechend signalisiert werden, da nicht bei jeder Abschaltung einer State Machine die Speicherung der Events sinnvoll ist und das System nicht entscheiden kann, ob eine State Machine endgültig deaktiviert wurde.

4.2.4 Schnittstellen

Es gibt sowohl eingehende wie auch ausgehende Schnittstellen, um die Kommunikation mit Fremdsystemen zu ermöglichen. Dies beinhaltet sowohl vom Fremdsystem durch Nachrichten ausgelöste Statusänderungen als auch bei Statusübergängen ausgelöste Aktionen oder Nachrichten an Fremdsysteme.

Eingehende Schnittstellen: Die eingehende Schnittstelle ermöglicht den Fremdsystemen das Melden einer Statusänderung an eine State Machine. Diese Nachricht soll möglichst einfach gehalten werden und nur die Identifikation der State Machine, den Event sowie allenfalls die Identifikation der Quelle und Parameter enthalten und soll so definiert sein, dass möglichst viele verschiedene Fremdsysteme damit kommunizieren können.

Ausgehende Schnittstellen: Für ausgehende Schnittstellen gelten grundsätzlich dieselben Vorgaben. Ausgelöste Aktionen können Nachrichten an Fremdsysteme sein oder ausgelöste Aktionen. Für Nachrichten an Fremdsysteme ist eine Schnittstelle mit denselben Vorgaben nötig wie für die eingehende Event Schnittstelle. Ausgelöste Aktionen können

Funktionslogik enthalten. Dies wird in *Modulare Schnittstellenerweiterungen* (auf dieser Seite) genauer erläutert.

Eine spezifische ausgehende Nachricht soll den Status eines Objekts an ein Fremdsystem weiterleiten. Diese Nachricht kann entweder als Aktion in einem Status definiert oder mittels Event abgefragt werden.

4.2.5 Modulare Schnittstellenerweiterungen

Um ausgehende Aktionen auszulösen kann es nötig sein, Funktionslogik in der Applikation integrieren zu müssen. Um daran im laufenden Betrieb Anpassungen oder Erweiterungen vornehmen zu können, sollen diese Teile modular durch eine Plug-In Infrastruktur eingebunden werden. So soll es möglich sein ein solches Modul zu ersetzen oder ein neues hinzuzufügen, ohne die gesamte Applikation zu beeinträchtigen.

4.2.6 UI für die Verwaltung

Zur Verwaltung der Deklarationen soll ein einfaches GUI erstellt werden. Für die Erstellung der Deklarationen enthält das GUI einen Editor, der das zusammenfügen der Zustände und Übergänge erleichtert. Die bekannten Events und Aktionen sollen übersichtlich aufgelistet werden, um sie den Zuständen beizufügen. Erstellte Deklarationen können im GUI auch verwaltet werden.

Als weitere Aufgabe können im GUI auch Events und Aktionen verwaltet werden. Die beinhaltet das Erstellen, Ändern und Löschen. Bei Aktionen gilt dabei die Einschränkung, dass die Funktion im System enthalten sein muss. Falls dies nicht der Fall ist, kann das System wie oben beschrieben mittels Plug-In ergänzt werden.

4.3 Nicht funktionale Anforderungen

4.3.1 Funktionalität

Richtigkeit

Fehlerhafte Ausgaben können im schlimmsten Fall lebensbedrohend sein, deshalb muss die Richtigkeit der Ausgaben stets gewährleistet sein.

Interoperabilität

Die Interoperabilität ist zu andern Systemen ist eine Kernaufgabe dieses Projekts. Fremde Systeme liefern Inputs oder dienen als Empfänger unserer Ausgaben. Um dies zu gewährleisten ist eine genaue Schnittstellendefinition nötig.

Sicherheit

Der Server auf welchem die State Machine läuft muss sich in einem Raum befinden, zu welchem nur ausgewählte Personen Zutritt haben. Die State Machines müssen vor Manipulation durch unberechtigte mittels Authentifizierung geschützt werden.

4.3.2 Zuverlässigkeit

Reife

Die maximal tolerierte Ausfallhäufigkeit und Ausfallzeit wird kundenspezifisch durch SLAs geregelt.

Wartbarkeit

Die Komponenten der Applikation sollen unabhängig voneinander gewartet werden können.

Wiederherstellbarkeit

Bei einem Systemabsturz soll der Betrieb schnellstmöglich wieder gewährleistet werden. Dies beinhaltet die Wiederherstellung des letzten Zustandes.

4.3.3 Benutzbarkeit

Verständlichkeit

Das System ist weitgehend selbsterklärend. Da dem Anwender nur die nötigen Elemente zu Verfügung gestellt werden, bleibt das System übersichtlich.

Bedienbarkeit

An wichtigen oder komplexen Stellen bietet das System unterstützende Informationen an.

4.3.4 Effizienz

Zeitverhalten

Die zu Überwachenden Objekte sind unterschiedlich Zeitkritisch. Deshalb soll das System wichtige oder dringende Objekte priorisieren können.

4.3.5 Änderbarkeit

Analysierbarkeit

Um mögliche Ursachen von aufgetretenen Fehlern analysieren zu können werden diese geloggt.

Modifizierbarkeit

Das Ziel dieses Projektes ist es eine Presence Engine zu entwickeln, welche sich leicht anpassen lässt.

Stabilität

Werden fehlerhafte Deklarationen einer State Machine geladen oder erreichen unbekannte Nachrichten das System, darf dies keine Auswirkungen auf andere Abläufe im System haben.

4.3.6 Übertragbarkeit

Anpassbarkeit

Das System soll modular aufgebaut sein, so dass die Teile unabhängig angepasst werden können.

Austauschbarkeit

Durch die oben erwähnte Modularität lassen sich einzelne Elemente austauschen.

4.3.7 Skalierbarkeit

Das System muss mindestens 100'000 aktive Objekte verwalten und betreiben können. Über die Anzahl der zu erwartenden Statusänderungen pro Sekunde liegen keine Angaben vor.

4.4 Rechtlich-vertragliche Anforderungen

4.4.1 Software

Diese Software bildet eine Komponente eines proprietären Systems, somit dürfen Fremdkomponenten nur verwendet werden, solange deren Lizenz ein einbinden in eine proprietären Software erlaubt.

4.4.2 Vertragliche Anforderungen

Es muss gemäss Vorlage auf der HSR-Website eine Vereinbarung über Urheber- und Nutzungsrechte von Auftraggeber, HSR und den Studierenden unterzeichnet werden.

4.5 Anforderungen an sonstige Lieferbestandteile

4.5.1 Software Dokumention

Die Dokumentation muss den Vorgaben für Diplom-, Bachelor- und Studienarbeiten der Abteilung Informatik entsprechen. Diese sind auf der HSR-Website hinterlegt. Alle Dokumente müssen bei Abgabe den Stand der Arbeit in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in vier Exemplaren abzugeben, sowie zwei Exemplare in gedruckter Form.

4.6 Use Cases

4.6.1 Use Case Model

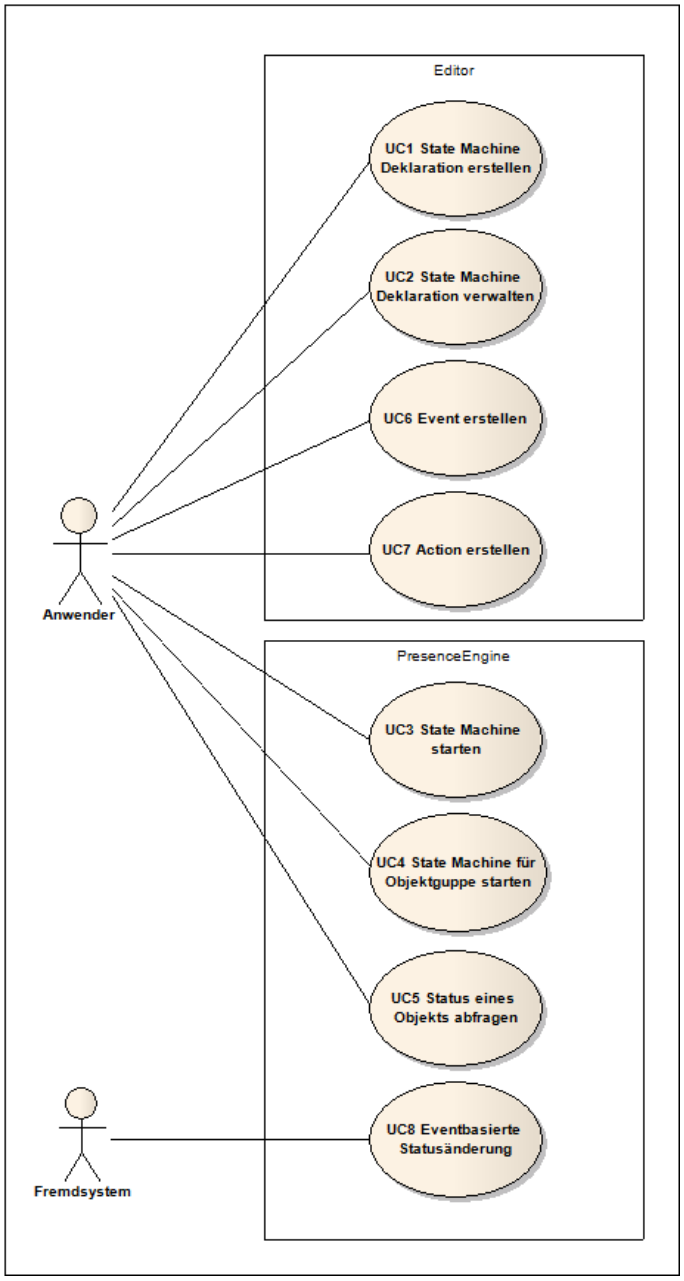


Abbildung (4.1) Use Case Model

4.6.2 Use Cases Brief

State Machine Deklaration Bearbeiten

Neue State Machine Deklaration erstellen: Der Anwender erstellt eine neue State Machine Deklaration. Dazu öffnet er den Editor und erfasst die States, deren Übergänge und Aktionen. Das System erstellt daraus ein State Machine Deklarationsfile, welches der Anwender speichern kann.

Bestehende State Machine Deklaration verwalten/verändern: Es sind kleinere Änderungen an einer State Machine nötig geworden. Der Anwender öffnet die Deklaration der zu ändernden State Machine mit dem Editor und passt diese an. Das System passt das Deklarationsfile an und der Anwender speichert die Deklaration wieder ab.

Neue State Machine Deklaration aus einer Bestehenden erstellen: Der Anwender will eine neue State Machine Deklaration auf Basis einer bestehenden erstellen. Hierfür lädt er eine Deklaration in den Editor und ändert sie ab. Das System erstellt daraus ein neues Deklarationsfile, welches der Anwender unter neuem Namen speichert.

CRUD State Machine

State Machine erstellen: Der Anwender weist die Presence Engine an, das Deklarationsfile zu laden. Das System öffnet das Deklarationsfile und erstellt die State Machine.

Alternativ kann eine State Machine auch automatisch erstellt werden. Das System bekommt den Event, dass ein neues Objekt zu überwachen ist. Anhand der Parameter wird die State Machine aus dem Deklarationsfile erstellt.

State Machine starten: Der Anwender weist das System an, eine neue Instanz einer State Machine zu erstellen, um ein Objekt zu überwachen. Das System erstellt die Instanz und startet sie.

State Machine unterbrechen: Der Anwender weist das System an, den Betrieb einer State Machine zu unterbrechen. Das System stoppt die State Machine und speichert sowohl die State Machine wie auch den aktuellen Status. Dies ermöglicht eine Wiederaufnahme des Betriebs dieser State Machine im selben Zustand.

State Machine deaktivieren: Der Anwender weist das System an, eine State Machine zu deaktivieren. Das System stoppt die State Machine und löscht diese.

State Machine für eine Objektgruppe erstellen: Die State Machine einer Gruppe wird durch den Anwender gleich erstellt wie die eines einzelnen Objektes. Danach kann er die Objekte der Gruppe hinzufügen unter der Voraussetzung, dass der Status des Objekts bereits mittels State Machine verfolgt wird.

Damit der Zustand der Gruppe überwacht werden kann, fügt das System den einzelnen State Machines der Objekte Funktionen hinzu. Diese Funktionen informieren die Gruppe über den Zustand ihrer Mitglieder.

Status eines Objektes abfragen

Der Anwender möchte den Status eines bestimmten Objektes wissen. Er gibt die Identifikation des Objekts ein und das System liefert den momentanen Status.

Event Bearbeiten

Neuen Event erstellen: Der Anwender will einen neuen Event definieren, der von einem Fremdsystem ausgelöst werden kann. Mit Hilfe des Editors erfasst er Name und Parameter des Events und speichert diesen ab.

Events verwalten/ändern: Der Anwender lädt einen bestehenden Event in den Editor und passt diesen an oder löscht ihn.

Installation einer neuen Erweiterung

Wird ein neues Modul für Aktionen oder Kommunikation mit Fremdsystemen benötigt, wird dieses vom Entwickler erstellt. Danach kann es als Plug-In ins bestehende System integriert werden.

Aktion Bearbeiten

Neue Aktion erstellen: Der Anwender will eine neue Aktion definieren, die aus einem Status ausgelöst werden kann. Mit Hilfe des Editors erfasst er Name und Parameter des Events und speichert diesen ab. Die Funktionalität muss dafür bereits im System vorhanden sein. Dafür muss allenfalls erst noch der obige Use Case (auf dieser Seite) ausgeführt werden.

Aktion verwalten/ändern: Der Anwender lädt eine bestehende Aktion in den Editor und passt diese an oder löscht sie.

Eventbasierte Statusänderung

Ein Fremdsystem sendet einen Event an die Presence Engine. Falls noch Events in Bearbeitung sind, wird der Event in die Eventqueue eingereiht. Sobald der Event an die

Reihe kommt gibt ihn die Presence Engine an die betroffene State Machine weiter, welche ihn verarbeitet. Entsprechend dem Event wird eine Statusänderung ausgelöst. Bei Austritt aus dem alten Status und Eintritt in den Neuen werden die entsprechenden Aktionen ausgelöst.

Priorisierte Behandlung eines Events: Ein Fremdsystem sendet einen Event an die Presence Engine, der priorisiert behandelt werden muss. Der Event wird in die Queue für priorisierte Events eingereiht, so dass er als nächstes bearbeitet wird. Die Presence Engine gibt den Event an die betroffene State Machine weiter, sobald er an die Reihe kommt. Danach erfolgt die Bearbeitung durch die State Machine wie bei normalen Events.

4.6.3 Use Cases Fully Dressed

State Machine Deklaration erstellen

Use Case ID:	UC1
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach und übersichtlich die State Machine gestalten und Events sowie Aktionen einbinden können.

Vorbedingungen

- Keine

Nachbedingungen

- Das Deklarationsfile der State Machine wurde erstellt und gespeichert.

Standardablauf

1. Der Anwender öffnet den State Machine Editor.
2. Der Anwender teilt dem System mit, dass er eine neue State Machine erstellen will.
3. Der Anwender gibt der State Machine einen Namen.
4. Der Anwender fügt der State Machine eine Status hinzu.

5. Der Anwender fügt dem Status einen Statusübergang hinzu.

   Schritt 5 für jeden Statusübergang wiederholen.

6. Der Anwender fügt dem Statusübergang eine Bedingung/Event hinzu.

   Schritt 6 für jeden Event wiederholen.

7. Der Anwender fügt dem Status eine Aktion hinzu.

   Schritt 7 für jede Aktion wiederholen.

   Schritte 4 bis 7 für jeden Status wiederholen.

8. Das System erstellt aus obiger Beschreibung das Deklarationsfile.
9. Der Anwender speichert das Deklarationsfile.

Erweiterungen

2a Der Anwender teilt dem System mit, dass er eine neue State Machine aus einer bestehenden erstellen will.

1. Der Anwender wählt ein bestehendes Deklarationsfile aus.
2. Das System lädt das Deklarationsfile in den Editor.

4a Der Anwender möchte eine State Machine als Sub State Machine hinzufügen.

1. Der Anwender wählt eine bestehendes State Machine aus.
2. Der Anwender fügt dem Eintrittspunkt zur Sub State Machine einen Statusübergang hinzu.

   Schritt 2 für jeden Statusübergang wiederholen.

3. Weiter im Standardablauf bei Schritt 6.

6a Die Bedingung/Event ist nicht bekannt.

1. Der Anwender erfasst den neuen Event mittels den Use Case (siehe Seite 55) .
2. Der Anwender fügt den Event dem Statusübergang hinzu.
3. Weiter im Standardablauf bei Schritt 6.

7a Die gewünschte Aktion ist nicht im System.

1. Weiter im Standardablauf bei Schritt 7.

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

State Machine Deklaration verwalten

Use Case ID:	UC2
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach die Deklarationen der State Machines verwalten können.

Vorbedingungen

- State Machine Deklarationen müssen gespeichert sein.

Nachbedingungen

- Das Deklarationsfile der State Machine wurde verändert und gespeichert oder gelöscht.

Standardablauf

1. Der Anwender öffnet den State Machine Editor.
2. Der Anwender teilt dem System mit, welche State Machine Deklaration er verändern möchte.
3. Das System lädt die entsprechende Deklaration.
4. Der Anwender nimmt Änderungen an der State Machine vor.
5. Der Anwender speichert das Deklarationsfile.

Erweiterungen

2a Der Anwender teilt dem System mit, welche State Machine Deklaration er löschen möchte.

1. Das System löscht das Deklarationsfile.

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

State Machine starten

Use Case ID:	UC3
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach die State Machine starten können.

Vorbedingungen

- State Machine Deklaration muss vorhanden sein.

Nachbedingungen

- Das Deklarationsfile der State Machine wurde in die Presence Engine geladen und die State Machine wurde gestartet.

Standardablauf

1. Der Anwender wählt die zu startende State Machine Deklaration aus.
2. Das System lädt die State Machine Deklaration.
3. Das System erstellt die State Machine.
4. Der Anwender benennt die Instanz der State Machine.
5. Der Anwender startet die State Machine.

Erweiterungen

1a Der Prozess wird mittels Event automatisch ausgelöst.

1. Das System lädt die im Event geforderte Deklaration.
2. Das System benennt die State Machine gemäss Anweisung aus Event.
3. Das System startet die State Machine.

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

State Machine für Objektgruppe starten

Use Case ID:	UC4
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach die State Machine für eine Gruppe starten können.

Vorbedingungen

- State Machine Deklaration muss vorhanden sein.

Nachbedingungen

- Das Deklarationsfile der State Machine wurde in die Presence Engine geladen und die State Machine wurde gestartet.
- Sämtliche zur Gruppe gehörigen Objekte wurden hinzugefügt.

Standardablauf

1. Der Anwender wählt die zu startende State Machine Deklaration aus.
2. Das System lädt die State Machine Deklaration.
3. Das System erstellt die State Machine.
4. Der Anwender benennt die Instanz der State Machine.
5. Der Anwender wählt das zur Gruppe gehörende Objekt aus.
6. Der Anwender fügt das ausgewählte Objekt der Gruppe hinzu.

   Schritte 6 und 7 für jedes zur Gruppe gehörende Objekt wiederholen.
7. Der Anwender startet die State Machine.

Erweiterungen

-

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

Status eines Objekts abfragen

Use Case ID:	UC5
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will wissen in welchem Status ein Objekt ist.

Vorbedingungen

- Das Objekt muss bekannt sein.
- State Machine muss gestartet sein.

Nachbedingungen

- Keine

Standardablauf

1. Der Anwender gibt die Referenz des gewünschten Objekts an.
2. Das System gibt den Status aus.

Erweiterungen

-

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Täglich mehrmals

Offene Punkte

- Keine

Event erstellen

Use Case ID:	UC6
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach einen neuen Event erstellen.

Vorbedingungen

- Keine

Nachbedingungen

- Der Neue Event wurde erstellt und gespeichert.

Standardablauf

1. Der Anwender öffnet den Editor.
2. Der Anwender teilt dem System mit, dass er einen neuen Event erstellen will.
3. Der Anwender benennt den Event.
4. Der Anwender füllt die weiteren Angaben für den Event aus.
5. Das System erstellt und speichert den Event.

Erweiterungen

2a Der Anwender teilt dem System mit, dass er einen bestehenden Event verändern will.

1. Der Anwender wählt einen bestehenden Event aus.
2. Der Anwender verändert und speichert den Event.

2b Der Anwender teilt dem System mit, dass er einen bestehenden Event löschen will.

1. Das System löscht den Event.

3a Der gewünschte Name ist bereits vergeben.

1. Das System informiert den Anwender, dass der Name vergeben ist.
2. Weiter im Standardablauf bei Schritt 3 .

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

Aktion erstellen

Use Case ID:	UC7
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	Anwender

Akteure und Interessen

Anwender Will möglichst einfach eine neue Aktion erstellen.

Vorbedingungen

- Die benötigte Logik für die Funktion ist im System vorhanden.

Nachbedingungen

- Die Neue Aktion wurde erstellt und gespeichert.

Standardablauf

1. Der Anwender öffnet den Editor.
2. Der Anwender teilt dem System mit, dass er eine neue Aktion erstellen will.
3. Der Anwender benennt die Aktion.
4. Der Anwender füllt die weiteren Angaben für die Aktion aus.
5. Das System erstellt und speichert die Aktion.

Erweiterungen

2a Der Anwender teilt dem System mit, dass er eine bestehende Aktion verändern will.

1. Der Anwender wählt eine bestehende Aktion aus.
2. Der Anwender verändert und speichert die Aktion.

2b Der Anwender teilt dem System mit, dass er eine bestehende Aktion löschen will.

1. Das System löscht die Aktion.

3a Der gewünschte Name ist bereits vergeben.

1. Das System informiert den Anwender, dass der Name vergeben ist.
2. Weiter im Standardablauf bei Schritt 3 .

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

Eventbasierte Statusänderung

Use Case ID:	UC8
Umfang:	Client
Ebene:	Anwenderziel
Primärakteur	System

Akteure und Interessen

-

Vorbedingungen

- Event ist bekannt.
- State Machine ist bekannt und aktiv.

Nachbedingungen

- Der Status wurde geändert.
- Sämtliche Aktionen wurden beim Übergang ausgelöst.

Standardablauf

1. Das System erhält einen Event von einem Fremdsystem.
2. Das System sucht die Ziel State Machine.
3. Das System leitet den Event an die richtige State Machine weiter.
4. Die State Machine löst die Statusänderung aus.
5. Die definierte Aktion wird ausgelöst.

   Schritt 4 für jede Aktion wiederholen.
6. Das Objekt wird in den neuen Status versetzt.

Erweiterungen

2a Die Ziel State Machine existiert nicht.

1. Der Event wird verworfen.

4a Der Event löst in dieser State Machine keine Statusänderung aus.

1. Der Event wird verworfen.

4b Der Event löst in diesem State keine Statusänderung aus.

1. Der Event wird verworfen.

Spezielle Anforderungen

- Keine

Abweichende Technologien und Daten

-

Auftretenshäufigkeit

Regelmässig

Offene Punkte

- Keine

5 Domain Analyse

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	28.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	28.03.2011	Reto Brühwiler	Domain Model
0.3	31.03.2011	Ricardo Alvarez	Überarbeitung gemäss Review
0.4	12.04.2011	Ricardo Alvarez	System Sequenzdiagramm und Contracts

5.1 Domain Model

5.1.1 Strukturdiagramm

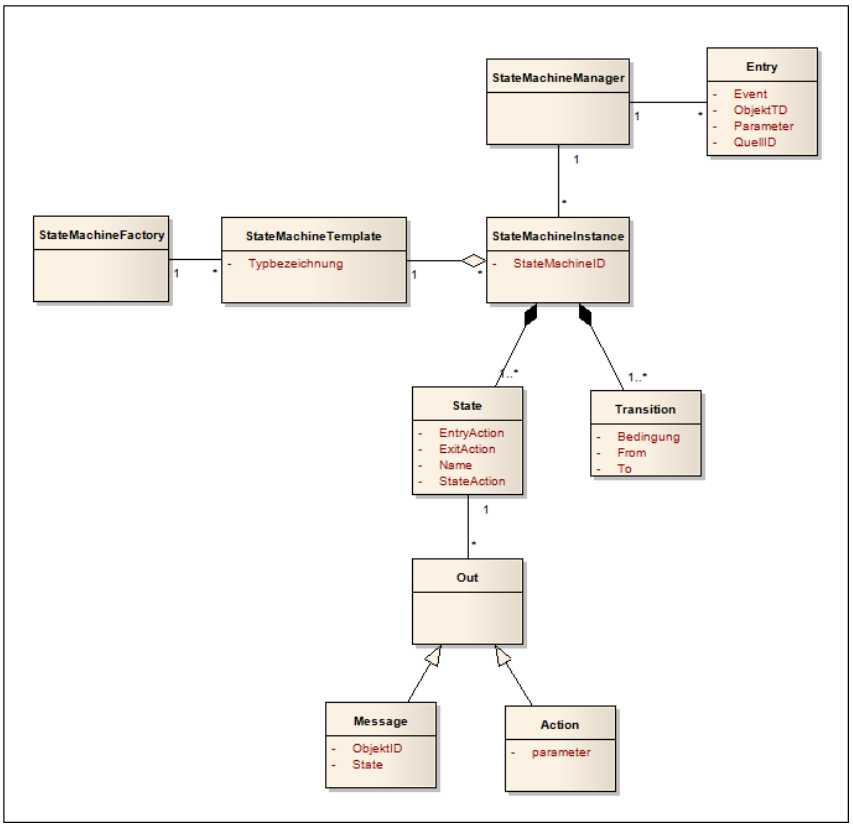


Abbildung (5.1) Domain Model

5.1.2 Konzeptbeschrieb

Das Strukurdiagramm zeigt die zum Erstellen, Verwalten und Betreiben deklarativer State Machines benötigten Komponenten und deren Zusammenhalt.

Mit Hilfe der StateMachineFactory werden Templates der State Machines (StateMachineTemplate) erstellt. Pro Objekt Kategorie wird ein Template erstellt. Für jedes Objekt dieses Typs wird eine Instanz (StateMachineInstance) eines solchen Templates erstellt.

Der StateMachineManager verwaltet diese State Machine Instanzen und leitet die Events aus Entry an die richtige Instanz weiter. Die daraus resultierenden Statusänderungen lösen Nachrichten (Message) oder Aktionen (Aktion) aus, welche an die Outputschnittstelle weiter gegeben werden.

5.1.3 Domain Objekte

Entry

Beschreibung

Entry bringt Werte der Inputschnittstelle in die benötigte Form für den StateMachineManager.

Attribute

ObjektID Eindeutige Identifikation des Objektes, welches durch die State Machine überwacht wird.

Event Event der an die State Machine übergeben wird.

QuellID Eindeutige Identifikation der Quelle, die den Event gesendet hat.

Parameter Parameter die mit dem Event übergeben werden.

Beziehungen

- StateMachineManager

Out

Beschreibung

Generiert Nachrichten oder Aktionen aus Statusänderungen und stellt diese in der gewünschten Form der Outputschnittstelle bereit.

Attribute

ObjektID Identifiziert das zu überwachende Objekt.

Status Beschreibt den Status des Objekts

Parameter Beinhaltet die Parameter welche einer Aktion mitgegeben werden.

Beziehungen

- State

State

Beschreibung

Repräsentiert einen Status in der State Machine.

Attribute

Name Name des Status.

EntryAction Aktionen, die bei Erreichen des States ausgeführt werden.

StateAction Aktionen, die innerhalb des States ausgeführt werden.

ExitAction Aktionen, die bei Verlassen des States ausgeführt werden.

Beziehungen

- StateMachineInstance
- Out

StateMachineFactory

Beschreibung

Erstellt ein State Machine Template aus der State Machine Description.

Attribute

-

Beziehungen

- StateMachineTemplate

StateMachineInstance

Beschreibung

Dies ist die eigentliche State Machine. Sie besteht aus States und Transitions und wird aus einem State Machine Template (StateMachineStructur) erzeugt. Verwaltet wird sie durch den StateMachineManager.

Attribute

StateMachineID Die ID der State Machine Instanz.

Beziehungen

- StateMachineManager
- StateMachineTemplate
- State(List)
- Transition(List)

StateMachineManager

Beschreibung

Der StateMachineManager verwaltet die State Machines. Er ist für das Weiterleiten der Events an die richtige State Machine Instanz verantwortlich.

Attribute

-

Beziehungen

- Entry(List)
- StateMachineInstance (List)

StateMachineTemplate

Beschreibung

Das StateMachineTemplate ist die Vorlage, aus welcher die einzelnen Instanzen der State Machines gebildet werden.

Attribute

Typbezeichnung Bezeichnet für welche Art Objekt dieses Template verwendet wird.

Beziehungen

- StateMachineFactory

Transition

Beschreibung

Repräsentiert einen Statusübergang in der State Machine.

Attribute

From Aus welchem State gewechselt wird.

To In welchen State gewechselt wird.

Bedingung Unter welchen Bedingung die Transaction durchgeführt wird.

Beziehungen

- StateMachineInstance

5.2 System Sequenzdiagramme

5.2.1 Systemoperationen

Use Case: Eventbasierte Statusänderung

Dieser Use Case deckt die Hauptaufgabe der Applikation ab, das Verarbeiten einkommender Events.

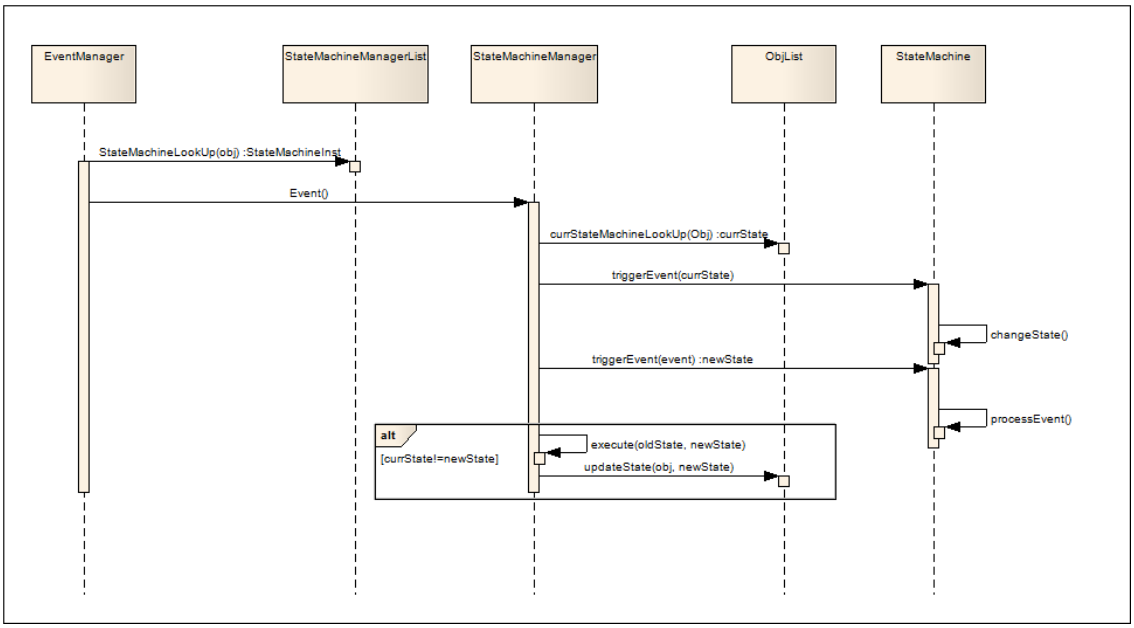


Abbildung (5.2) Use Case: Eventbasierte Statusänderung

5.2.2 Operation Contracts

Contracts Use Case: Eventbasierte Statusänderung

Operation Contract 1

Name:	StateMachineLookUp(obj)
Vorbedingung:	Das Objekt ist im System erfasst.
Nachbedingung:	Der StateMachineManager des Objekts ist bekannt.

Tabelle (5.1) OC1

Operation Contract 2

Name:	Event()
Vorbedingung:	Der StateMachineManager des Objekts ist bekannt.
Nachbedingung:	Der Event wurde an den richtigen StateMachineManager weitergeleitet.

Tabelle (5.2) OC2

Operation Contract 3

Name:	currStateLookUp(obj)
Vorbedingung:	-
Nachbedingung:	Der momentane Status des Objekts ist bekannt.

Tabelle (5.3) OC3

Operation Contract 4

Name:	triggertEvent(currState)
Vorbedingung:	-
Nachbedingung:	Die StateMachine ist im Ausgangsstatus des Objekts.

Tabelle (5.4) OC4

Operation Contract 5

Name:	triggertEvent(event)
Vorbedingung:	Die StateMachine ist im Ausgangsstatus des Objekts.
Nachbedingung:	Der Event wurde verarbeitet.

Tabelle (5.5) OC5

Operation Contract 6

Name:	execute(oldState, newState)
Vorbedingung:	Eine Statusänderung auf dem Objekt hat stattgefunden.
Nachbedingung:	Die Aktionen wurden ausgeführt.

Tabelle (5.6) OC6

Operation Contract 7

Name:	updateState(obj, newState)
Vorbedingung:	Eine Statusänderung auf dem Objekt hat stattgefunden.
Nachbedingung:	Der neue Status des Objekts wurde gespeichert.

Tabelle (5.7) OC7

5.3 Activity Diagramm

Das folgende Activity Diagramm zeigt den Ablauf einer Event Verarbeitung.

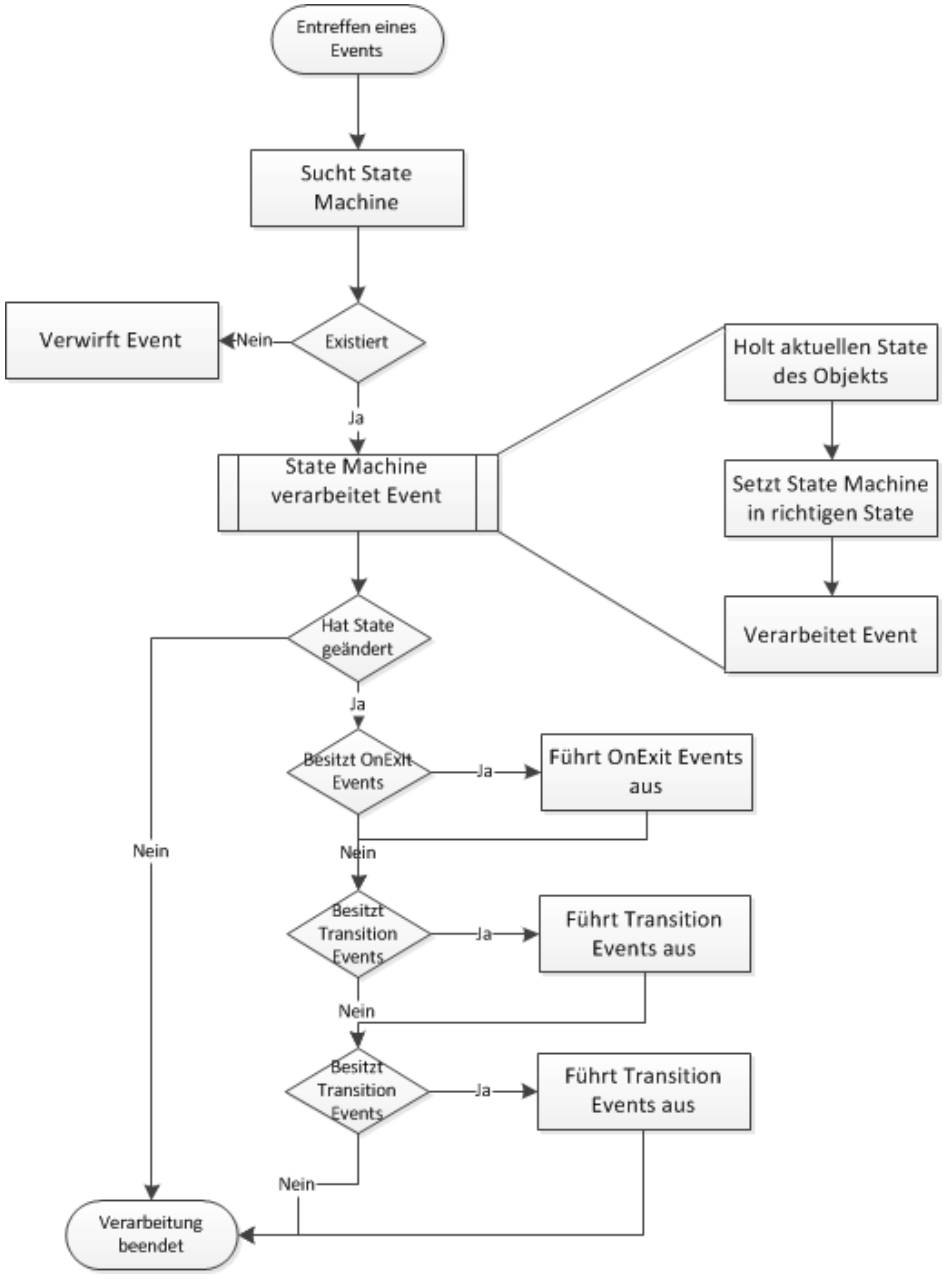


Abbildung (5.3) Activity Diagram

6 Externes Design

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	08.06.2011	Reto Brühwiler	Dokument erstellt

6.1 Navigation Map

Dieses Kapitel beschreibt die Navigationsmöglichkeiten innerhalb der Benutzeroberflächen. Beim Starten der Silverlightwebsite wird der Gruppen Tab der Homeansicht geladen. Mittels Tab kann zur Verwaltungsansicht der Deklarationen gewechselt werden. Die restlichen Ansichten sind als Silverlight Child Window implementiert und werden durch Buttons geöffnet.

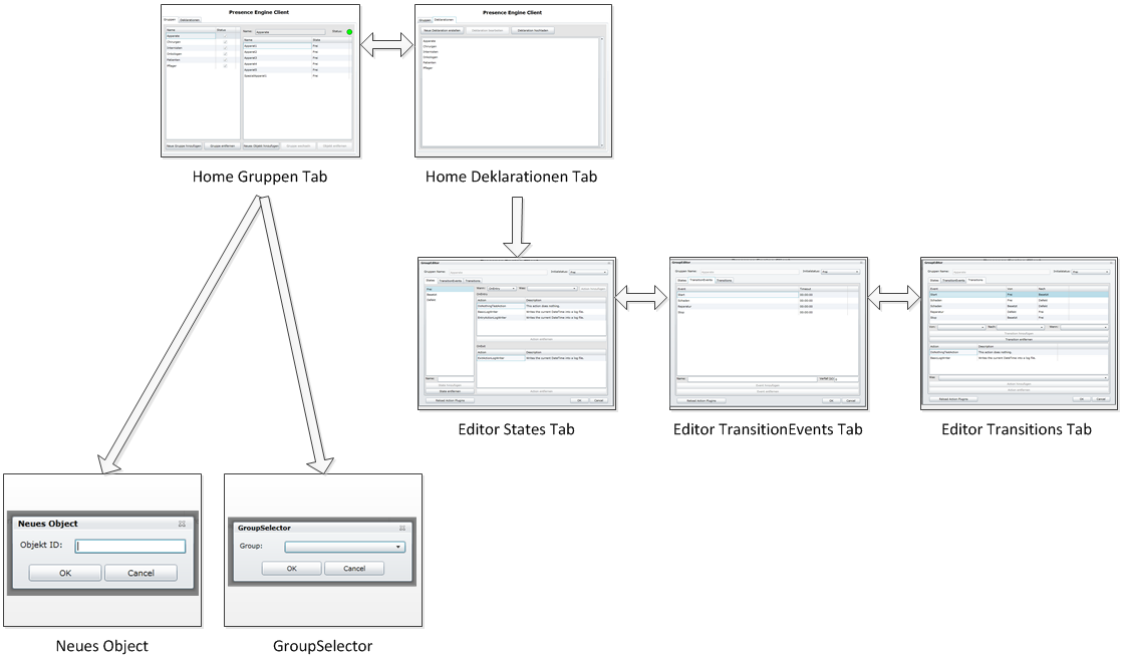


Abbildung (6.1) Navigation Map

6.2 UI

6.2.1 Home

Abbildung 6.2 zeigt die Startseite, auf welcher die Gruppen und deren zu überwachende Objekte dargestellt werden. Es können neue Gruppen und Objekte erstellt und bestehende gelöscht werden.

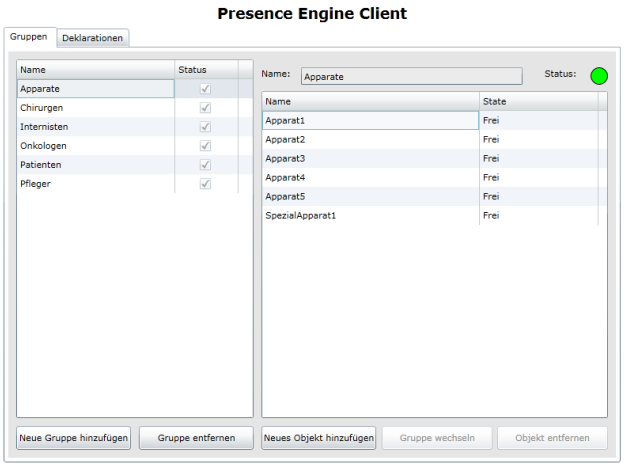


Abbildung (6.2) Gruppen Tab

In diesem Tab werden sämtliche auf dem Server gespeicherten Deklarationen aufgelistet. Es können neue hinzugefügt oder bestehende angepasst werden.

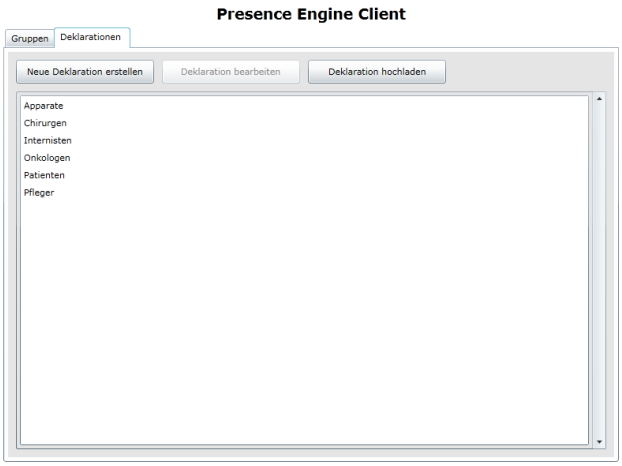


Abbildung (6.3) Deklarationen Tab

6.2.2 Objekt Gruppen

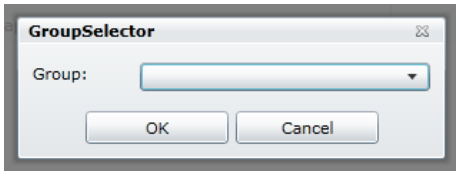


Abbildung (6.4) Objekt Gruppen Auswahl

6.2.3 Neues Objekt hinzufügen

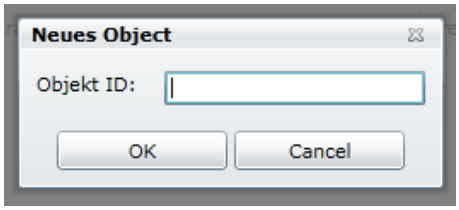


Abbildung (6.5) Neues Objekt

6.2.4 Editor

Mit Hilfe des Editors können neue Deklarationen erfasst und bestehende verändert werden. Er besteht aus drei Tabs. Im ersten Tab werden die States und deren Actions konfiguriert, im zweiten die Events und im dritten die Transitions mit den dazugehörigen Actions.

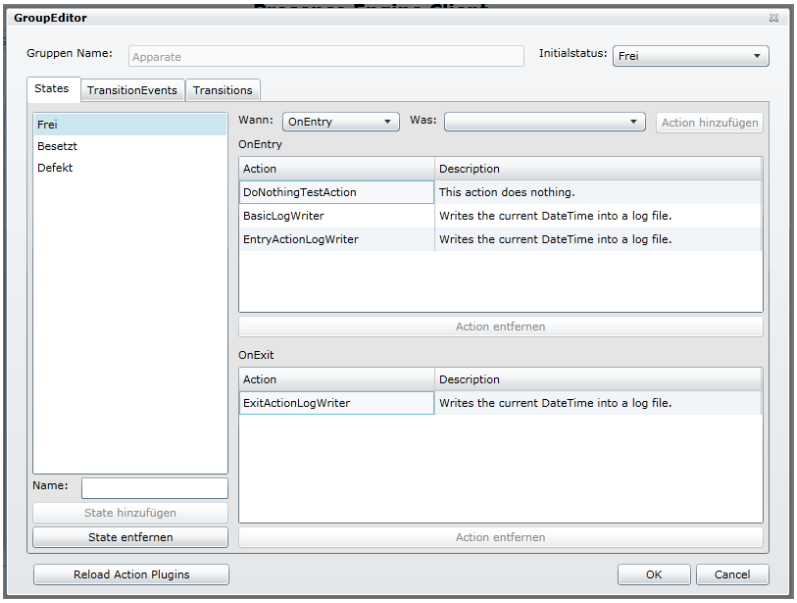


Abbildung (6.6) Editor State Tab

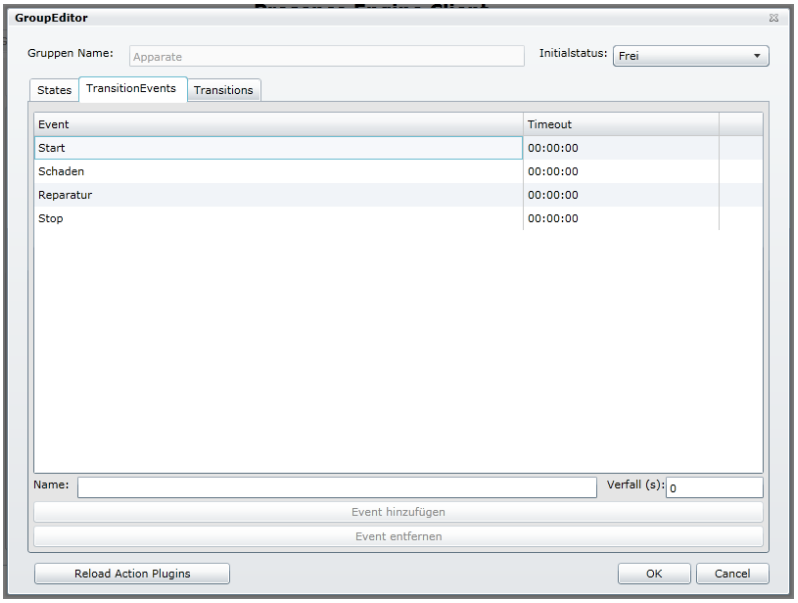


Abbildung (6.7) Editor TransitionEvents Tab

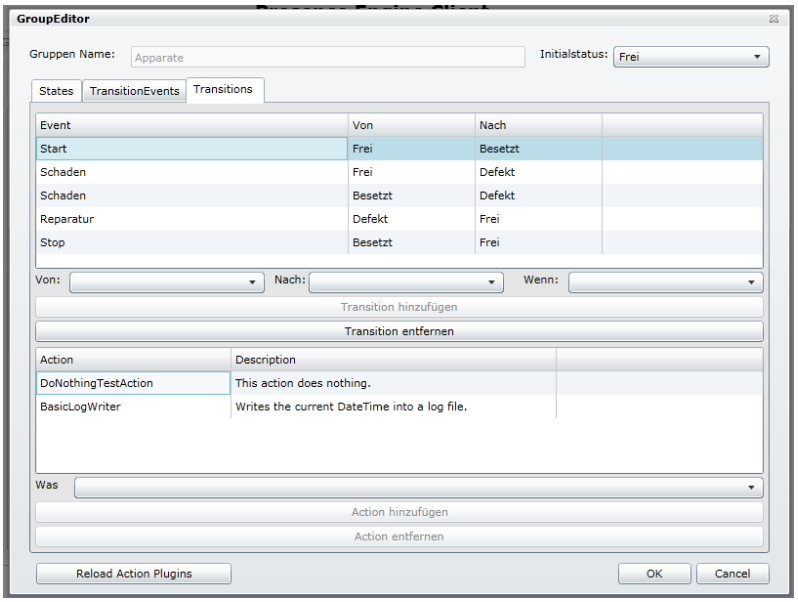


Abbildung (6.8) Editor Transitions Tab

7 Software Architektur Dokument

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	30.05.2011	Ricardo Alvarez	Dokument erstellt
0.2	30.05.2011	Ricardo Alvarez	Architektonische Entscheidungen
0.3	02.06.2011	Ricardo Alvarez	Architekturkonzepte
0.4	07.06.2011	Ricardo Alvarez	Logische Architektur, Implementierung
0.5	09.06.2011	Reto Brühwiler	MVVM, Datenspeicherung

7.1 Umgebung

Ausgangslage für das Projekt ist der im Abschnitt *Überblick* (siehe Seite 3) der Problemstellung erläuterte Wunsch, Objekte und deren Zustand im Spitalumfeld zu verwalten.

Das folgende Bild gibt einen Überblick über die bisherige Umsetzung im Spitalumfeld.

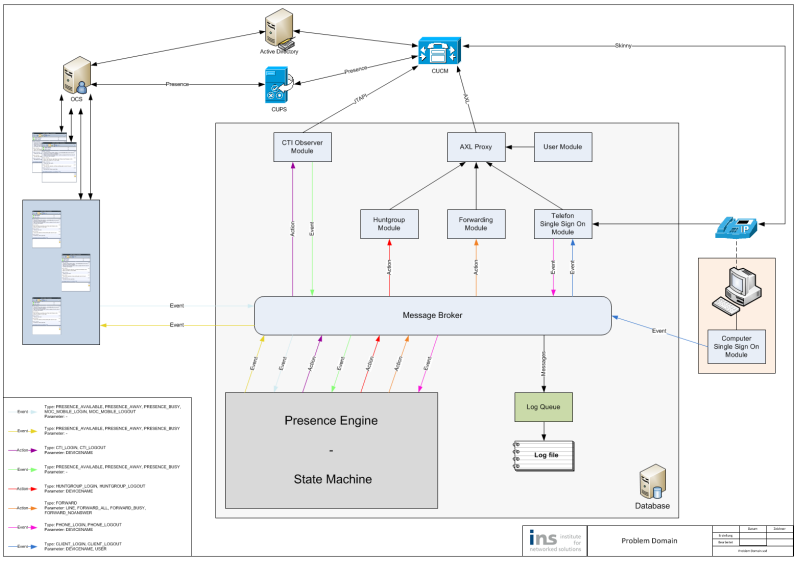


Abbildung (7.1) Überblick Implementation Spital

Dies entwickelte sich in dieser Arbeit dahingehend weiter, dass die von uns umgesetzte Lösung nicht speziell auf das Spitalumfeld angepasst ist, sondern den Rahmen bietet, um einfach ins gewünschte Umfeld eingepasst zu werden. Dies gewährleistet die deklarative und modulare Umsetzung.

Das folgende Bild gibt einen Überblick über die Systemgrenze der Presence Engine.

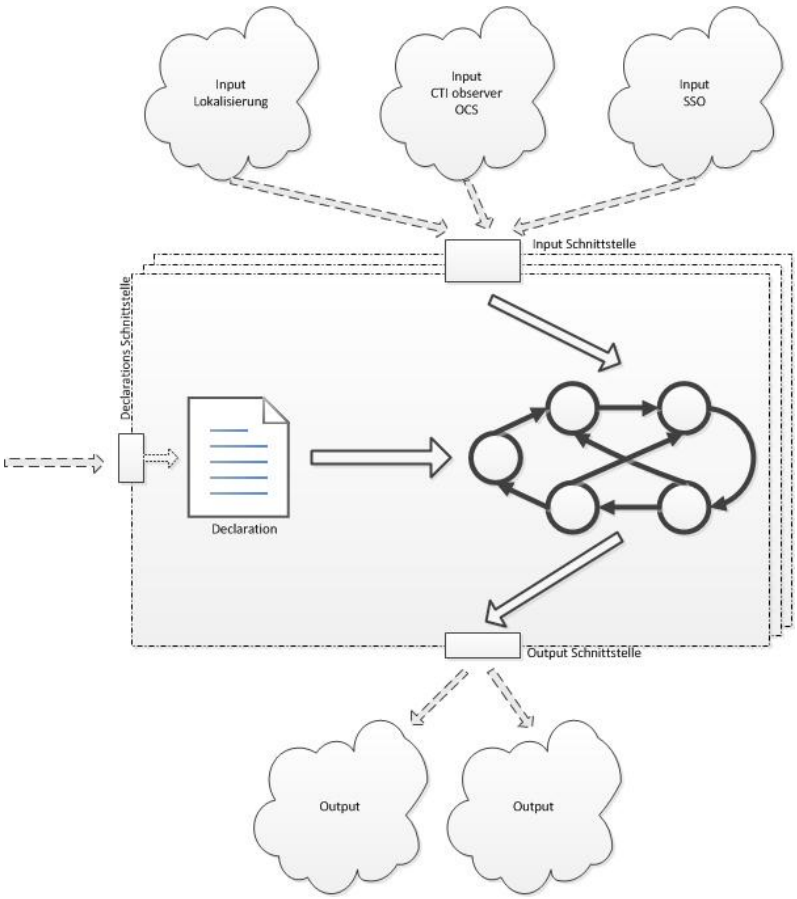


Abbildung (7.2) Überblick Systemgrenze

7.2 Architektonische Ziele

Wichtigstes Ziel dieser Arbeit war die Flexibilität und leichte Adaptierbarkeit der Applikation auf ein neues Umfeld. Dies soll durch die modulare Erweiterbarkeit sowie deklarative Beschreibung ermöglicht werden.

7.3 Architektonische Entscheidungen

7.3.1 UI-Architektur

Wie kann die Logik des Programms vom User Interface getrennt werden? Wie wird eine saubere Trennung der Schichten erreicht?

Faktoren

- Lose Kopplung.
- Saubere Schichtentrennung.

Lösung

MVVM[1](Model, View, ViewModel) ist der Standard für die Umsetzung des MVC Patterns in .Net und C# und speziell auf diese Umgebung ausgelegt. Es ist sowohl für die Benutzung mit WPF als auch mit Silverlight vorgesehen.

Ungelöste Probleme

Keine.

Erwogene Alternativen

Keine.

7.3.2 State Machine

Die State Machine bildet den Kern der Presence Engine. Sie ist zuständig für das Verarbeiten ankommender Events. Dabei muss die Vorgabe berücksichtigt werden, dass sehr viele Objekte zu verwalten und viele Events in kurzer Zeit zu verarbeiten sind.

Wie kann eine effiziente und schlanke State Machine implementiert werden? Wie kann die Verwaltung so vieler Objekte realisiert werden?

Faktoren

- Speichereffizienz der statusbehafteten Objekte.
- Verarbeitungseffizienz der Events.

1 http://msdn.microsoft.com/de-de/magazine/dd419663.aspx

Lösung

Aufgrund der Untersuchung bestehender Technologien im Kapitel *State of the Art* (siehe Seite 7) hat sich das Team für die Verwendung von Microsoft .Net und der .bbv Common State Machine entschied. Danach wurde in *Architektur Analyse* (siehe Seite 19) die Architektur der State Machine Verwaltung ermittelt. Die Ergebnisse obiger Analysephase haben das Team veranlasst, nur eine State Machine pro Objektgruppe zu erstellen. Die Voraussetzung einer Objektgruppe ist dabei, dass alle enthaltenen Objekte dieselbe Deklaration der State Machine besitzen.

Ungelöste Probleme

Keine.

Erwogene Alternativen

Die Hauptalternative war, je Objekt eine State Machine zu erstellen. Dafür hätte die effizientere Verarbeitung gesprochen. Dagegen allerdings der höhere Speicherverbrauch, wie in *Architektur Analyse* (siehe Seite 19) beschrieben ist.

7.3.3 Deklaration der State Machine

Um die State Machine im Deklarationsfile zu beschreiben, soll eine einfache und bekannte Beschreibungssprache genutzt werden.

Faktoren

- Einfache Beschreibung der State Machine im Deklarationsfile.
- Bekannte Sprache zur Beschreibung nutzen.

Lösung

Das Team hat sich zur Verwendung von XML als Beschreibungssprache der State Machine entschieden, weil XML sowohl einfach zu nutzen als auch gut bekannt und weit verbreitet ist. Des Weiteren ist die Beschreibung in XML auch für Menschen lesbar, was ein weiterer Vorteil ist. So können auch Deklarationen ausserhalb dieses Systems erstellt und eingelesen werden, was der Applikation mehr Flexibilität gibt.

Ungelöste Probleme

Keine.

Erwogene Alternativen

Wir haben diverse Workflow Beschreibungssprachen angeschaut, da diese für von uns untersuchte Frameworks vorgesehen sind zum Beschreiben der Workflows. Dies beinhaltete fologende Spezifikationssprachen:

BPMN 2.0 Die Business Process Modeling Notation[1], welche von OMG[2] (Object Management Group) herausgegeben wird.

Yawl Yawl steht für Yet Another Workflow Language[3] und ist eine Spezifikationssprache aus dem universitären Umfeld. Ziel von Yawl ist es, Workflow Beschreibungen mathematisch überprüfbar darzustellen.

XAML Die Windows Workflow Foundation benutzt zur Beschreibung der Abläufe XAML[4] (Extensible Application Markup Language).

Allen diesen Sprachen gemein ist, dass sie zur Darstellung komplexer Businessprozesse ausgelegt sind und somit viel mehr Möglichkeiten bieten als wir zur Beschreibung einer State Machine benötigen. Dies macht die Erstellung solcher Beschreibungen nur unnötig Kompliziert, vorallem wenn kein Editor verwendet wird.

7.3.4 Erweiterbarkeit der Aktionen.

Anforderung an die Presence Engine ist es in einem State automatisch Funktionen auslösen zu können, zum Beispiel bei Eintritt, Übertritt oder Austritt eines States. Da bei Inbetriebnahme des Systems möglicherweise noch nicht alle benötigten Funktionen bekannt sind, sollen diese zur Laufzeit hinzugefügt werden können. So soll ein modular erweiterbares System von Aktionen entstehen.

Faktoren

- Einfache Erweiterung um weitere Aktionen.
- Einbinden der Erweiterungen zur Laufzeit.

Lösung

Microsoft .Net bietet seit der Version 4 mit MEF[5] (Managed Extensibility Framework) eine Bibliothek die es ermöglicht, Erweiterungen zur Laufzeit zu laden. Dank der standardisierten Infrastruktur erleichtert MEF das Erstellen der Erweiterungen. Das Framework

1 http://de.wikipedia.org/wiki/Business_Process_Modeling_Notation
2 http://www.omg.org
3 http://www.yawlfoundation.org
4 http://msdn.microsoft.com/de-de/library/ms752059.aspx
5 http://msdn.microsoft.com/de-de/library/ee332203.aspx

bietet die Ermittlung der zu ladenden Typen und instanziiert diese. So wird auch das Einbinden der Erweiterungen sehr einfach gehalten.

Ungelöste Probleme

Keine.

Erwogene Alternativen

Das ältere MAF[1] (Managed Add-In Framework) ist bereits seit .Net 3.5 Teil des Frameworks. Das oben beschriebene MEF ist einfacher und leichtgewichtiger als MAF. Der Vorteil von MAF ist, dass die Erweiterungen besser von der Hauptanwendung getrennt werden können. Da in unserem Fall die Erweiterungen aber durch die Entwickler der Presence Engine geschrieben werden, kommt der Vorteil von MAF nicht zum Tragen.

7.3.5 Event Schnittstelle

Die Schnittstelle, um Events in die Presence Engine einzuspeisen soll von allen Fremdsystemen benutzt werden können. Dies beinhaltet sowohl die zu Beginn bekannten wie auch später hinzukommende Systeme. Somit muss die Schnittstelle möglichst offen definiert sein um von allen erreicht werden zu können.

Faktoren

- Offene Schnittstelle.
- Universell ansprechbar.

Lösung

Eine Web Service Schnittstelle ist universell und von allen Fremdsystemen ansprechbar. Idee hinter dieser Lösung ist, dass sich hinzukommende Systeme welche mit unserer Applikation kommunizieren wollen unsere Schnittstelle ansprechen. Dies hat den Hintergrund, dass zu Beginn nicht bekannt ist, was für weitere Systeme später hinzukommen können. Dies gewährleistet wiederum die Flexibilität und Erweiterbarkeit unserer Applikation.

Ungelöste Probleme

Keine.

1 http://msdn.microsoft.com/de-de/library/bb384200.aspx

Erwogene Alternativen

Als Alternative zu einer Web Service Schnittstelle wurde in Erwägung gezogen, verschiedene spezialisierte Schnittstellen zu erstellen. Dies hat aber den Nachteil, dass später eventuell neue Schnittstellen benötigt werden, welche nicht im System vorhanden sind. Eine modulare Erweiterbarkeit der Schnittstelle konnte nicht sinnvoll umgesetzt werden, weil die Erweiterungsmöglichkeiten zu wenig umfassend waren, als dass es einen Vorteil gegenüber Web Service gehabt hätte.

7.4 Architekturkonzepte

7.4.1 MVVM Model-View-ViewModel Pattern

Ziel des MVVM ist es, das Design von den Funktionalitäten zu trennen. Die Gestaltung der Elemente wird in der XAML Deklaration der Views festgelegt. Die Views sollen möglichst kein Code-Behind enthalten. So wird erreicht, dass die UIs leicht ausgetauscht werden können. Dadurch kann der Designer die UIs gestalten ohne ständiges einbeziehen des Entwicklers. Die Views beziehen die benötigten Daten durch Bindings in der XAML Deklaration vom ViewModel. Das ViewModel beinhaltet die Daten für die Views sowie deren Funktionalität, kennt den Inhalt der Views jedoch nicht. Die Daten bezieht das ViewModel von den Models, welche die Verbindung zu gespeicherten Daten repräsentiert.

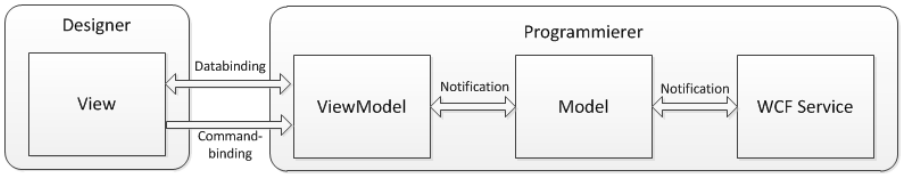


Abbildung (7.3) MVVM Schichten

Das Binding der View an das ViewModel wird mittels DataContext Property im Code-Behind festgelegt. Dies sollte möglichst der einzige Code-Behind sein.

Die Funktionalitäten der Buttons werden durch Bindings an die im ViewModel definierten ICommands realisiert und benötigen somit kein Code-Behind. Jeder dieser ICommands besitze eine CanExecute und eine Execute Funktion. Die CanExecute Funktion gibt einen Booleanwert zurück, ist dieser False kann der Button nicht verwendet werden. In der XAML Deklaration kann dem Button noch ein CommandParameter mitgegeben werden. Ändert sich dieser Parameter wird die CanExecute Funktion ausführt und somit erneut überprüft, ob der Button verwendet werden darf. Die Execute Funktion wird ausgeführt wenn der Button gedrückt wird.

Durch die vom MVVM geforderten Trennung lassen sich die UI Funktionalitäten einzeln testen, durch simulieren von Userinteraktionen.

7.4.2 Deklaration der State Machine

Das Team hat sich zur Verwendung von XML als Beschreibungssprache der State Machine entschieden, weil XML sowohl einfach zu nutzen als auch gut bekannt und weit verbreitet ist. Des Weiteren ist die Beschreibung in XML auch für Menschen lesbar, was ein weiterer Vorteil ist. So können auch Deklarationen ausserhalb dieser Applikation erstellt werden, was der Applikation mehr Flexibilität gibt. Nachfolgend soll ein Überblick über Struktur und Elemente einer Deklaration gegeben werden.

XML-Struktur

Die folgende Seite zeigt das Schema der XML-Struktur:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/
  2001/XMLSchema">
  <xs:element name="statemachine">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="states">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="state">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="actions">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="entryactions" type="xs:string" />
                          <xs:element name="exitactions" type="xs:string" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="transitevents">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="transitevent">
                <xs:complexType>
                  <xs:attribute name="eventname" type="xs:string" use="required" />
                  <xs:attribute name="eventtimeoutduration" type="xs:unsignedByte" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="transitions">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="transition">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="actions">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="transitionactions">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element maxOccurs="unbounded" name="transitionaction" type="xs:string"
  />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="from" type="xs:string" use="required" />
                  <xs:attribute name="to" type="xs:string" use="required" />
                  <xs:attribute name="on" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="initialstate" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Abbildung (7.4) Das XML Schema der Deklaration

Elemente der Deklaration

RootElement: Im folgenden Bild ist das Grundgerüst der XML-Deklaration abgebildet:

```
<?xml version="1.0" encoding="utf-8"?>
<statemachine initialstate="InitialStateName">
  <states>

  </states>
  <transitevents>

  </transitevents>
  <transitions>

  </transitions>
</statemachine>
```

Abbildung (7.5) Das Grundgerüst der Deklaration mit Root-Element

Das Root-Element der Deklaration ist <statemachine>. Das Attribut dieses Elements, der initialstate, bezeichnet den ersten State der Machine, also der Zustand in dem sich ein Objekt nach dem Hinzufügen zur State Machine als erstes befindet.

Das Root-Element enthält drei Elemente wie in obigem Beispiel zu sehen ist. Dies sind die Elemente für States, TransitEvents und Transitions.

Definition eines States: Der unten abgebildete State Knoten ist ein Beispiel für die Definition eines Zustands.

```
<state id="Statename">
  <actions>
    <entryactions>Actionname</entryactions>
    <exitactions>Actionname</exitactions>
  </actions>
</state>
```

Abbildung (7.6) State Knoten

Das State Element hat als Attribut die ID des States. Im Kindknoten <actions> können die zu diesem Zustand gehörenden Aktionen definiert werden. Dabei ist zu unterscheiden zwischen Aktionen die bei Eintritt und jenen die bei Austritt des Zustands ausgelöst werden sollen. Der Name der aufzurufenden Funktion wird als Inhalt des entsprechenden Elements eingetragen.

Definition eines TransitEvents: Der unten abgebildete TransitEvent Knoten ist ein Beispiel für die Definition eines Übergangsevents.

```
<transitevent eventname="Eventname" eventtimeoutduration="300"/>
```

Abbildung (7.7) TransitEvent Knoten

Das Element TransitEvent hat zwei Attribute. Das erste ist der Name des Events, das zweite die Lebensdauer des Events in Sekunden, wobei eine 0 für eine unbegrenzte Lebensdauer steht.

Definition einer Transition: Der unten abgebildete Transition Knoten ist ein Beispiel für die Definition eines Zustandsübergangs.

```
<transition from="Statename" to="Statename" on="Eventname">
  <actions>
    <transitionactions>
      <transitionaction>Actionname</transitionaction>
    </transitionactions>
  </actions>
</transition>
```

Abbildung (7.8) Transition Knoten

Die drei Attribute haben folgende Bedeutung:

- from: enthält die ID des Ausgangszustands.
- to: enthält die ID des Zielzustands.
- on: enthält den Namen des TransitEvents, der diesen Übergang auslöst.

Desweiteren enthält auch die Transition einen Kindknoten für Aktionen wie der State. Die enthaltenen Kindknoten sind vom Typ <transitionaction> und enthalten den Namen der Funktion, die bei dieser Transition aufgerufen werden soll.

Wichtig bei der Deklaration ist, dass die in der Transition benutzten States und TransitionEvents in den oben beschriebenen Knoten definiert sind. Ansonsten kann die Deklaration nicht geladen werden.

Beispiel einer Deklaration

Das folgende Bild zeigt ein Beispiel einer solchen Deklaration:

```xml
<?xml version="1.0" encoding="utf-8"?>
<statemachine id="Apparate" initialstate="Frei">
  <states>
    <state id="Frei">
      <actions>
        <entryactions/>
        <exitactions/>
      </actions>
    </state>
    <state id="Besetzt">
      <actions>
        <entryactions/>
        <exitactions/>
      </actions>
    </state>
  </states>
  <transitevents>
    <transitevent eventname="Start" eventtimeoutduration="0"/>
    <transitevent eventname="Stop" eventtimeoutduration="0"/>
  </transitevents>
  <transitions>
    <transition from="Frei" to="Besetzt" on="Start">
      <actions>
        <transitionactions>
          <transitionaction>DoNothingTestAction</transitionaction>
          <transitionaction>BasicLogWriter</transitionaction>
        </transitionactions>
      </actions>
    </transition>
    <transition from="Besetzt" to="Frei" on="Stop">
      <actions>
        <transitionactions>
          <transitionaction/>
        </transitionactions>
      </actions>
    </transition>
  </transitions>
</statemachine>
```

Abbildung (7.9) Beispiel einer State Machine Deklaration

7.4.3 Plugin Infrastruktur für Aktionen

Die Plugin Infrastruktur wurde mit Hilfe des Managed Extensibility Frameworks, kurz MEF, implementiert. MEF ist eine Bibliothek, die Programmstrukturen zur Implementation der Erweiterbarkeit zur Laufzeit bietet und Bestandteil des Microsoft .Net 4 Frameworks. Es stellt die Ermittlung von ladbaren Typen, die Instanziierung und das Zusammensetzten der Plugins in der Kernapplikation zur Verfügung.

Konzept

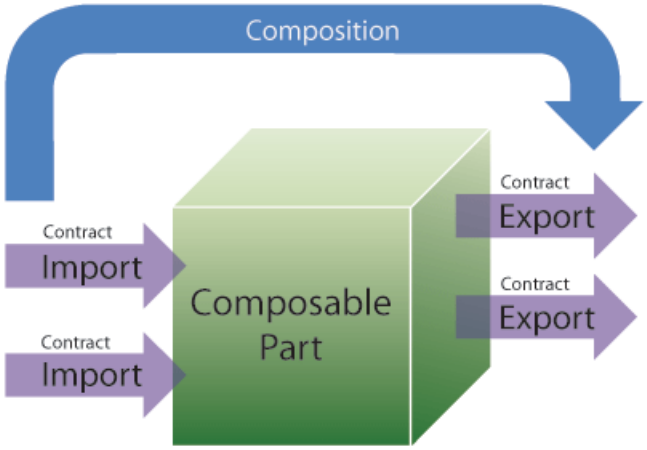


Abbildung (7.10) Konzepte von MEF

Das Konzept von MEF beruht auf folgenden Bestandteilen:

Composable Part Dies sind die Bestandteile von MEF, welche Exports und Imports zur Verfügung stellen oder laden. Ein Beispiel für ein Composable Part ist das Plugin, welches geladen wird.

Export Ein Export kennzeichnet einen Service, der geladen werden kann.

Import Ein Import kennzeichnet die Funktion, welche die Exports laden kann.

Contract Der Contract legt die Regeln fest für den Import eines Teils. Das Interface IActionPlugin, welches alle benötigten Felder für den Import definiert ist in unserer Applikation der Contract.

Composition Dies beschreibt den Vorgang des Zusammensetzens aller Importierten Teile.

Die Applikation, welche Plugins laden will, muss die Komposition der Teile auslösen. MEF sucht dann selbständig an den spezifizierten Orten nach importierbaren Bestandteilen,

also Klassen oder Funktionen, welche als Export gekennzeichnet sind. Die importierten Teile werden dann in einem Katalog bereitgestellt.

Das folgende Bild zeigt einen Überblick der Architektur von MEF:

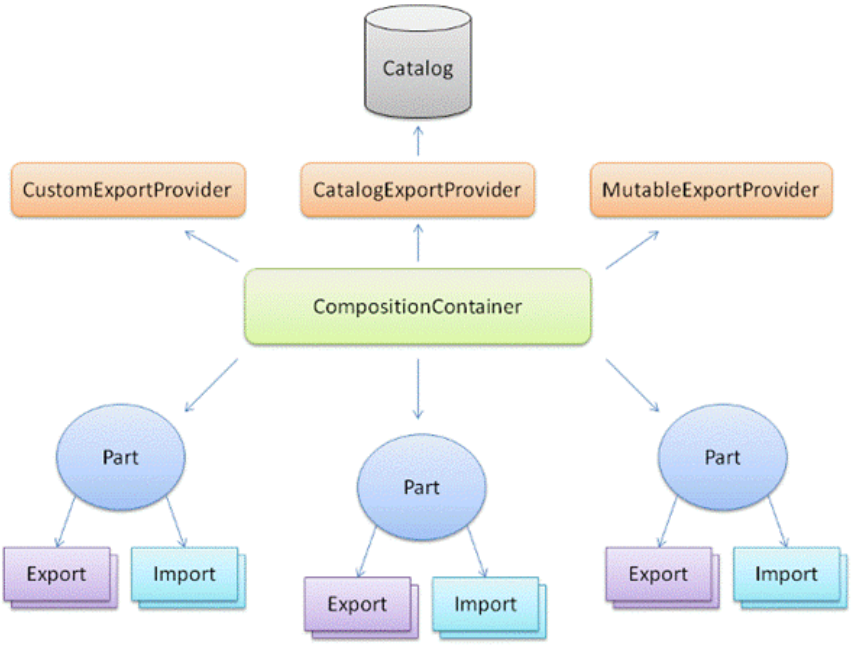


Abbildung (7.11) Vereinfachte Architektur von MEF

Plugins erstellen

Um neue Plugins zu erstellen muss im eigenen Projekt die Bibliothek (siehe Seite 107) eingebunden werden. Diese enthält das Interface IActionPlugin, welches durch das Plugin implementiert werden muss.

IActionPlugin Das Interface enthält folgende drei Felder, die zwingend implementiert werden müssen im Plugin:

ActionName Der Name der Aktion. Dieser muss in der Presence Engine einzigartig sein, damit eine eindeutige Zuteilung möglich ist. Ist dies nicht der Fall, wird die zweite Aktion mit selbem Namen in der Presence Engine nicht zur Verfügung gestellt.

ActionDescription Die Beschreibung der Aktion. Diese wird im Editor angezeigt, zur einfacheren Benützung.

Action Die eigentlich Action. Also die Funktion, die durch die Presence Engine ausgeführt wird.

Die Klasse im Plugin, welche das Interface implementiert, muss mit Attribut Export gekennzeichnet werden. Ausser dieser Kennzeichnung und dem Interface ist nicht weiter nötig von seiten des Plugin Entwicklers. Das folgende Bild zeigt eine Beispiel Implementation eines Action Plugins:

```
[Export(typeof(IActionPlugin))]
class TestActionPlugIn : IActionPlugin
{
    public string ActionName
    {
        get { return "TestAction"; }
    }

    public string ActionDescription
    {
        get { return "Dies ist eine Testfunktion."; }
    }

    public Action<string> Action
    {
        get { return TheTestAction; }
    }

    private void TheTestAction(string objectID)
    { }
}
```

Abbildung (7.12) Beispiel Implementation eines Action Plugin

7.4.4 Event Schnittstelle

Wie beschrieben ist die Event Schnittstelle als Web Service implementiert, damit alle Fremdsysteme Events an die Presence Engine liefern können.

WSDL der Eventschnittstelle

Wie der Schnittstelle Events geliefert werden können beschreibt das WSDL auf der nächsten Seite.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions name="EventService" targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:tns="http://tempuri.org/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
    xmlns:wsa10="http://www.w3.org/2005/08/addressing"
    xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  - <wsdl:types>
    - <xsd:schema targetNamespace="http://tempuri.org/Imports">
        <xsd:import schemaLocation="http://localhost:11000/EventService.svc?
          xsd=xsd0" namespace="http://tempuri.org/" />
        <xsd:import schemaLocation="http://localhost:11000/EventService.svc?
          xsd=xsd1"
          namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
      </xsd:schema>
    </wsdl:types>
  - <wsdl:message name="EventService_EnterEvent_InputMessage">
      <wsdl:part name="parameters" element="tns:EnterEvent" />
    </wsdl:message>
  - <wsdl:message name="EventService_GetObjectState_InputMessage">
      <wsdl:part name="parameters" element="tns:GetObjectState" />
    </wsdl:message>
  - <wsdl:message name="EventService_GetObjectState_OutputMessage">
      <wsdl:part name="parameters" element="tns:GetObjectStateResponse" />
    </wsdl:message>
  - <wsdl:message name="EventService_AddNewObjectToGroup_InputMessage">
      <wsdl:part name="parameters" element="tns:AddNewObjectToGroup" />
    </wsdl:message>
  - <wsdl:message name="EventService_DeleteObjectFromGroup_InputMessage">
      <wsdl:part name="parameters" element="tns:DeleteObjectFromGroup" />
    </wsdl:message>
  - <wsdl:message name="EventService_ChangeGroupMembership_InputMessage">
      <wsdl:part name="parameters" element="tns:ChangeGroupMembership" />
    </wsdl:message>
  - <wsdl:portType name="EventService">
    - <wsdl:operation name="EnterEvent">
        <wsdl:input wsaw:Action="http://tempuri.org/EventService/EnterEvent"
          message="tns:EventService_EnterEvent_InputMessage" />
      </wsdl:operation>
    - <wsdl:operation name="GetObjectState">
        <wsdl:input wsaw:Action="http://tempuri.org/EventService/GetObjectState"
          message="tns:EventService_GetObjectState_InputMessage" />
        <wsdl:output
          wsaw:Action="http://tempuri.org/EventService/GetObjectStateResponse"
          message="tns:EventService_GetObjectState_OutputMessage" />
      </wsdl:operation>
    - <wsdl:operation name="AddNewObjectToGroup">
        <wsdl:input
          wsaw:Action="http://tempuri.org/EventService/AddNewObjectToGroup"
          message="tns:EventService_AddNewObjectToGroup_InputMessage" />
      </wsdl:operation>
    - <wsdl:operation name="DeleteObjectFromGroup">
```

```
      <wsdl:input
        wsaw:Action="http://tempuri.org/EventService/DeleteObjectFromGroup"
        message="tns:EventService_DeleteObjectFromGroup_InputMessage" />
    </wsdl:operation>
  - <wsdl:operation name="ChangeGroupMembership">
      <wsdl:input
        wsaw:Action="http://tempuri.org/EventService/ChangeGroupMembership"
        message="tns:EventService_ChangeGroupMembership_InputMessage" />
    </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="EventService" type="tns:EventService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="EnterEvent">
      <soap:operation soapAction="http://tempuri.org/EventService/EnterEvent"
        style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  - <wsdl:operation name="GetObjectState">
      <soap:operation
        soapAction="http://tempuri.org/EventService/GetObjectState"
        style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="AddNewObjectToGroup">
      <soap:operation
        soapAction="http://tempuri.org/EventService/AddNewObjectToGroup"
        style="document" />
    + <wsdl:input>
    </wsdl:operation>
  - <wsdl:operation name="DeleteObjectFromGroup">
      <soap:operation
        soapAction="http://tempuri.org/EventService/DeleteObjectFromGroup"
        style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  - <wsdl:operation name="ChangeGroupMembership">
      <soap:operation
        soapAction="http://tempuri.org/EventService/ChangeGroupMembership"
        style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="EventService">
  - <wsdl:port name="EventService" binding="tns:EventService">
      <soap:address location="http://localhost:11000/EventService.svc" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Schnittstelle

Über die Schnittstelle können folgende Funktionen ausgelöst werden:

EnterEvent Mit dieser Funktion können neue Events der Presence Engine übergeben werden. Wie ein Event aufgebaut ist, wird im Kapitel 7.4.4 (auf dieser Seite) beschrieben. Die vier benötigten Parameter sind als Strings definiert und müssen der Funktion als solche übergeben werden. Wenn der Objektidentifikator und der Eventbezeichner dem System bekannt sind wird der Event verarbeitet und führt zu Statusänderungen und allenfalls ausgelösten Aktionen. Ansonsten wird der eingehende Event verworfen. Der Erhalt der Nachricht wird nicht an die Quelle bestätigt und es werden auch keine Fehlermeldungen an die Quelle weitergeleitet.

GetObjectState Das Aufrufen dieser Funktion liefert den momentanen Zustand des Objekts, sofern die Objekt ID im System bekannt ist. Dies erlaubt es einem Fremdsystem den gegenwärtigen Zustand zu erfragen. Zum Beispiel kann so ermittelt werden, ob eine Person die durch das System überwacht wird gerade frei ist.

Object-CRUD Die drei Funktionen AddNewObjectToGroup, DeleteObjectFromGroup und ChangeGroupMembership dienen der Automatisierung der Objektverwaltung. Sie erlauben es einem Fremdsystem ein neues Objekt zu erstellen und einer Objektgruppe hinzuzufügen, es zu ändern oder zu löschen. So kann der Prozess der Objektverwaltung automatisiert werden und flexibel auf Änderungen reagiert werden.

Event

Ein Event, wie er in der Presence Engine verarbeitet wird, hat die unten beschriebenen Felder und kann wie oben beschrieben über den Web Service in der Presence Engine ausgelöst werden.

eventname Der Name des Events. Anhand dieses Bezeichners wird in der State Machine der Übergang ausgelöst.

objectidentifier Der Bezeichner des Objekts, für das der Event ausgelöst wurde.

priority Die Verarbeitungspriorität des Events. Dieses Feld kann zwei Werte annehmen: NORMAL und URGENT.

timestamp Datum und Uhrzeit, zu dem das Ereignis erzeugt wurde in UTC nach ISO 8601. Folgendes Beispiel zeigt das Zeitformat: 2011-06-03T15:08:36.

7.5 Logische Architektur

Die Applikation besteht aus mehreren Layern. Dazu gehören der Presentation Layer, Business Layer und Data Access Layer.

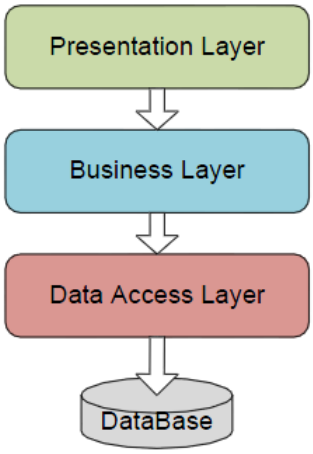


Abbildung (7.13) Layer Architektur

7.5.1 Übersicht

Die Solution der Presence Engine Applikation besteht aus sieben Projekten. Diese sind nachfolgend alphabetisch Aufgelistet und kurz beschrieben.

PresenceEngine.ActionPluginContract Dieses Projekt enthält das Interface, welches zur Erstellung von Action Plugins gebraucht wird.

PresenceEngine.Common In diesem Projekt sind alle gemeinsamen Ressourcen enthalten, wie Interfaces und Exceptions.

PresenceEngine.Core Dieses Projekt bildet den Kern der Applikation. Es enthält die Verarbeitungslogik für Events und die benötigten Klassen für Erstellen und Verwalten der State Machines und Gruppen.

PresenceEngine.Persistence Dieses Projekt enthält die Logik des Data Access Layer.

PresenceEngine.Service In diesem Projekt sind die Services definiert, welche zur Kommunikation gebraucht werden.

PresenceEngine.UI In diesem Projekt befindet sich das User Interface.

PresenceEngine.Test Dieses Projekt enthält die Unit-Tests.

7.5.2 Übersicht der Package- und Klassenstruktur

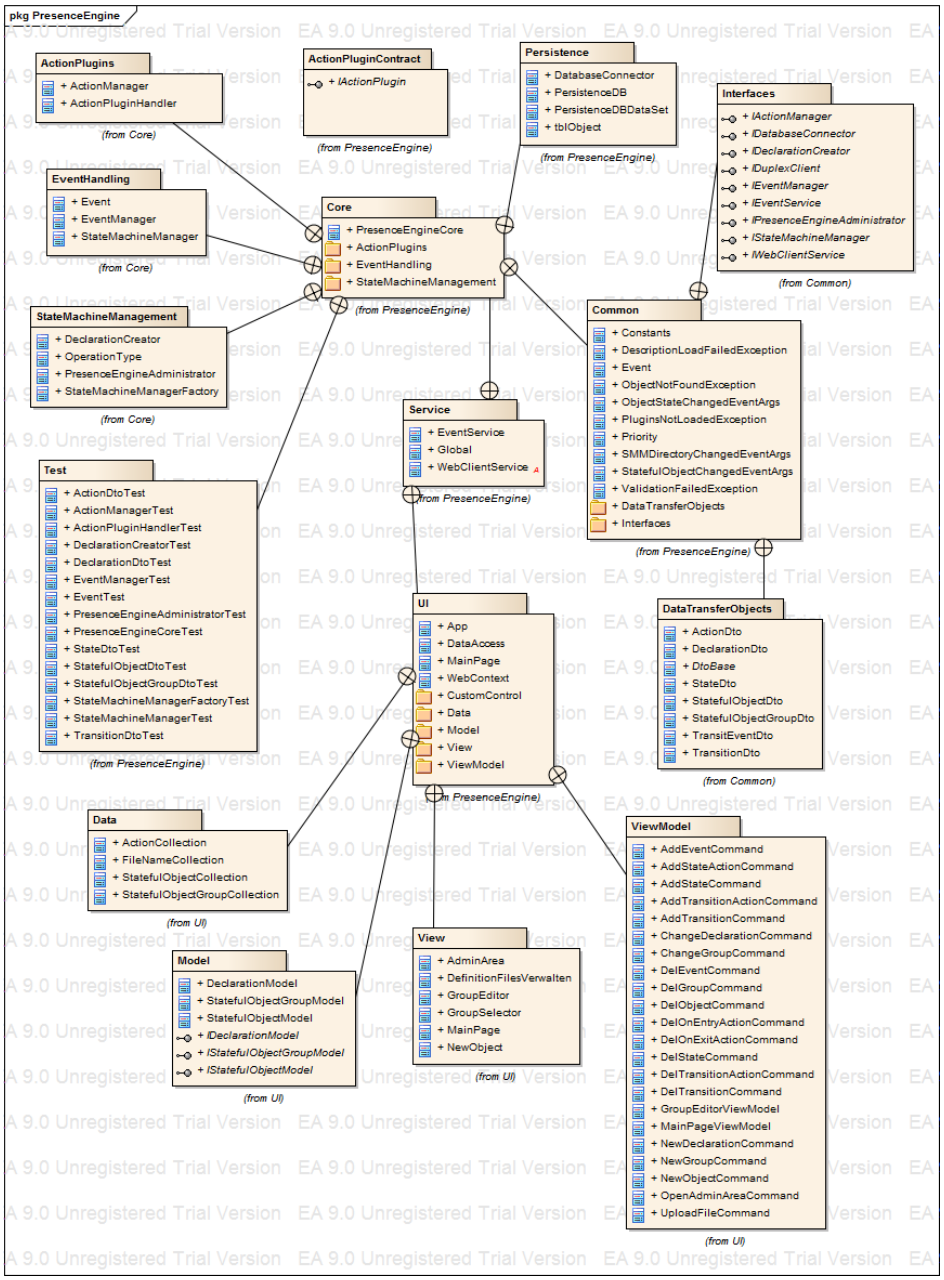


Abbildung (7.14) Package- und Klassenstruktur

7.5.3 Package-Abhängigkeiten

Das folgende Bild zeigt die Package-Abhängigkeiten, wie sie durch NDepend dargestellt werden.

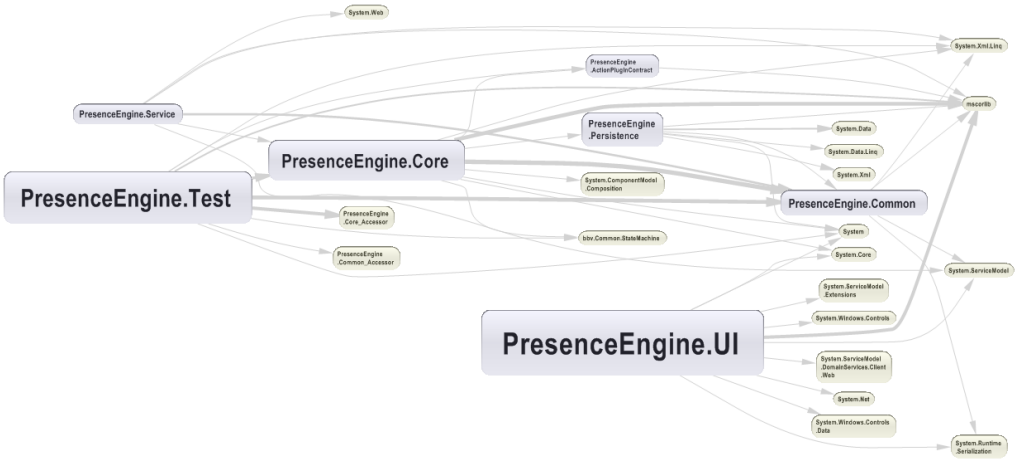


Abbildung (7.15) Package-Abhängigkeiten

7.5.4 Design Pakete

Package PresenceEngine.ActionPluginContract

In diesem Projekt ist nur das Interface enthalten für die Erstellung weiterer Plugins. Die resultierende Klassenbibliothek muss in anderen Projekten eingebunden werden, so dass die Plugins in die Presence Engine eingebunden werden können.

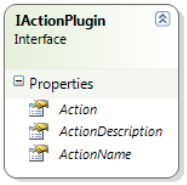


Abbildung (7.16) Klassendiagramm ActionPluginContract

Klassenname	Beschreibung
IActionPlugin	Das in Plugins zu implementierende Interface.

Tabelle (7.1) Package PresenceEngine.ActionPluginContract

PresenceEngine.Common

Das Projekt PresenceEngine.Common enthält Hilfsklassen für die ganze Applikation. Dies beinhaltet die Transfer Objekte für die Kommunikation mit dem User Interface, welche als DataContract durch den Service gesandt werden müssen, sowie die Definition von Interfaces und Exceptions.

Abbildung (7.17) Klassendiagramm PresenceEngine.Common

Klassenname	Beschreibung
Constants	Enthält alle Konstanten der Applikation.
Event	Der zu verarbeitende Event.
DescriptionLoadFailedException	Fehlermeldung, wenn eine Deklaration nicht geladen werden kann.
ObjectNotFoundException	Fehlermeldung, wenn ein Objekt nicht im System vorhanden ist.
PluginsNotLoadedException	Fehlermeldung für Probleme beim Laden von Plugins.
ValidationFailedException	Fehlermeldung für die Validierung.
StatefulObjectChangedEventArgs	Argumente für den Event wenn ein statusbehaftetes Object ändert.
SMMDirectoryChangedEventArgs	Argumente für den Event wenn es Änderungen im State Machine Manager Directory gibt.

Tabelle (7.2) Package PresenceEngine.Common

PresenceEngine.Common.DataTransferObjects In diesem Package sind die Datenobjekte für die Kommunikation mit dem UI enthalten.

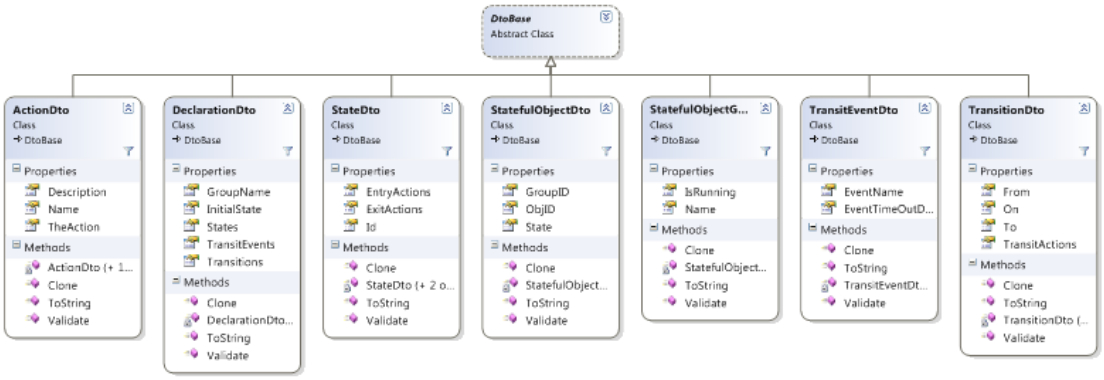


Abbildung (7.18) Klassendiagramm PresenceEngine.Common.DataTransferObjects

Klassenname	Beschreibung
ActionDto	Transfer Objekt für eine Action.
DeclarationDto	Transfer Objekt für die Deklaration einer Objektgruppe.
DtoBase	Die Basisklasse der Transfer Objekte.
StateDto	Transfer Objekt für einen Zustand.
StatefulObjectDto	Transfer Objekt für ein statusbehaftetes Objekt.
StatefulObjectGroupDto	Transfer Objekt für einer Gruppe von Objekten.
TransitEventDto	Transfer Objekt für einen TransitEvent.
TransitionDto	Transfer Objekt für einen Zustandsübergang.

Tabelle (7.3) Package PresenceEngine.Common.DataTransferObjects

PresenceEngine.Common.Interfaces Hier befinden sich die Interfaces für Services und Daten Access.

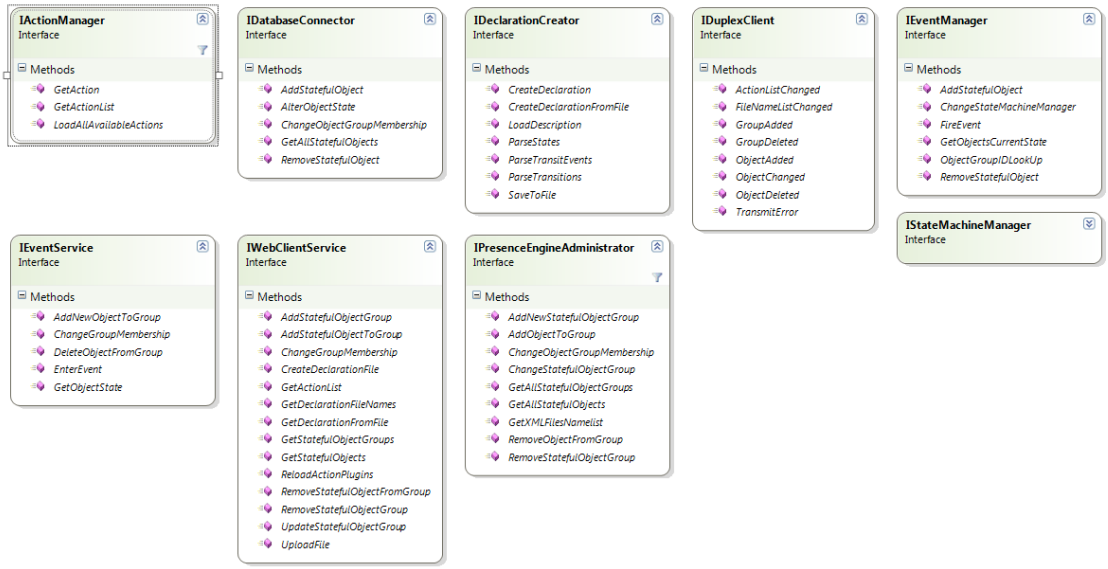


Abbildung (7.19) Klassendiagramm PresenceEngine.Common.Interfaces

Klassenname	Beschreibung
IActionManager	Das Interface, um den ActionManager zu nutzen.
IDatabaseConnector	Das Interface für den Data Access Layer.
IDeclarationCreator	Das Interface, um die Klasse DeclarationCreator zu nutzen.
IEventManager	Das Interface, um den EventManager zu nutzen.
IPresenceEngineAdministrator	Das Interface, um die Klasse PresenceEngineAdministrator zugänglich zu machen.
IEventService	Dieses Interface enthält den ServiceContract für die Eventschnittstelle.
IStateMachineManager	Hilfs-Interface für IEventManager.
IWebClientService	Dieses Interface enthält den ServiceContract für den WebClientService.

Tabelle (7.4) Package PresenceEngine.Common.Interfaces

PresenceEngine.Core

Dies ist das Hauptprojekt der Presence Engine und enthält die Applikationslogik.

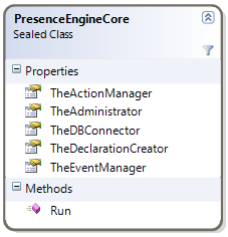


Abbildung (7.20) Klassendiagramm PresenceEngine.Core

Klassenname	Beschreibung
PresenceEngineCore	Hier wird der Kern der Applikation durch den Service gestartet.

Tabelle (7.5) Package PresenceEngine.Core

PresenceEngine.Core.ActionPlugins

In diesem Package sind die Klassen für das Handling der Action-Plugins und das Verwalten der Aktionen.

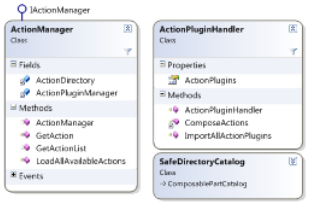


Abbildung (7.21) Klassendiagramm PresenceEngine.Core.ActionPlugins

Klassenname	Beschreibung
ActionManager	Diese Klasse ist für das Verwalten der Aktionen zuständig.
ActionPluginHandler	Ist Zuständig für das Laden der Plugins.

Tabelle (7.6) Package PresenceEngine.Core.ActionPlugins

PresenceEngine.Core.Eventhandling Dieses Package enthält die für die Verarbeitung der Events und Verwaltung der State Machines benötigten Klassen.

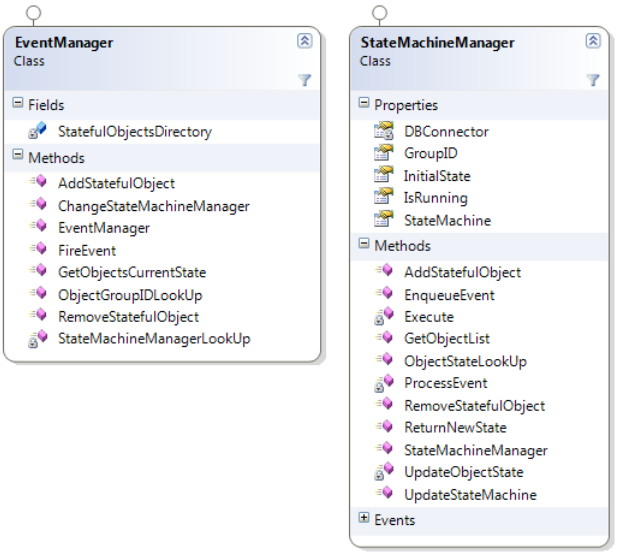


Abbildung (7.22) Klassendiagramm PresenceEngine.Core.Eventhandling

Klassenname	Beschreibung
EventManager	Verwaltet die Objektliste und leitet einkommende Events an die richtige State Machine weiter zur Verarbeitung.
StateMachineManager	Diese Klasse veraltet die Objektgruppen und enthält die State Machine. Events werden hier von der State Machine Verarbeitet und Aktionen bei Bedarf ausgelöst.

Tabelle (7.7) Package PresenceEngine.Core.StateMachineManagement

PresenceEngine.Core.StateMachineManagement Hier befinden sich die für die Verwaltung der Presence Engine benötigten Klassen.

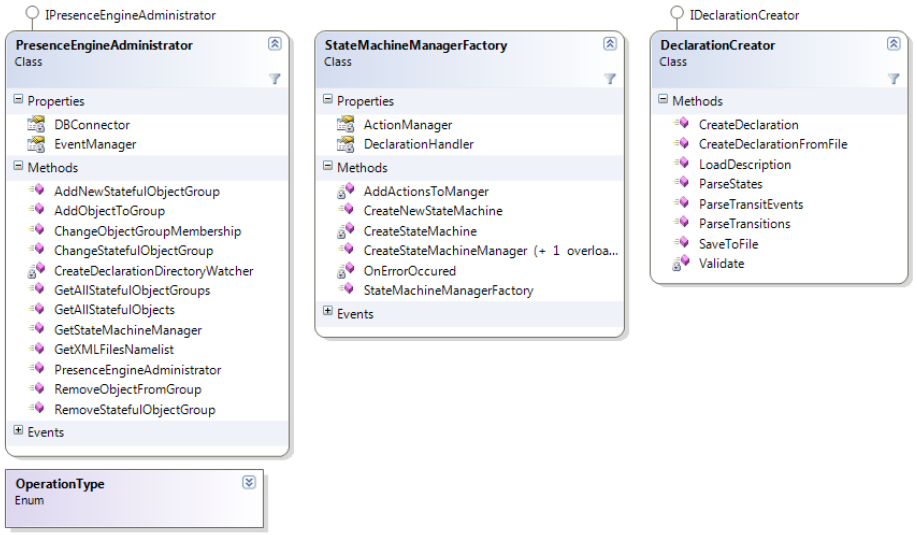


Abbildung (7.23) Klassendiagramm PresenceEngine.Core.StateMachineManagement

Klassenname	Beschreibung
DeclarationCreator	Erstellt, schreibt und liest Deklarationen von State Machines.
PresenceEngineAdministrator	Ist Zuständig für die Verwaltung der Presence Engine. Dies beinhaltet das Erstellen neuer Deklarationen sowie die Instanziierung von Objektgruppen und deren Objekten.
StateMachineManagerFactory	Erstellt neue Objektgruppen und deren State Machine.

Tabelle (7.8) Package PresenceEngine.Core.StateMachineManagement

PresenceEngine.Persistence

Dieses Projekt enthält die Logik des Data Access Layer.

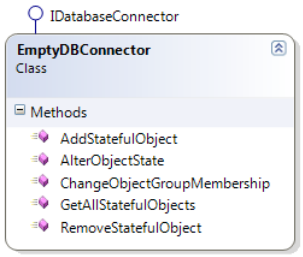


Abbildung (7.24) Klassendiagramm PresenceEngine.Persistence

Klassenname	Beschreibung
DatabaseConnector	Diese Klasse stellt die Verbindung zur Datenbank.

Tabelle (7.9) Package PresenceEngine.Persistence

PresenceEngine.Service

Dieses Projekt startet die Applikation in dem der Service auf dem IIS gestartet wird. Die beiden enthaltenen Services bilden die Schnittstelle zu den Fremdsystemen und dem UI.

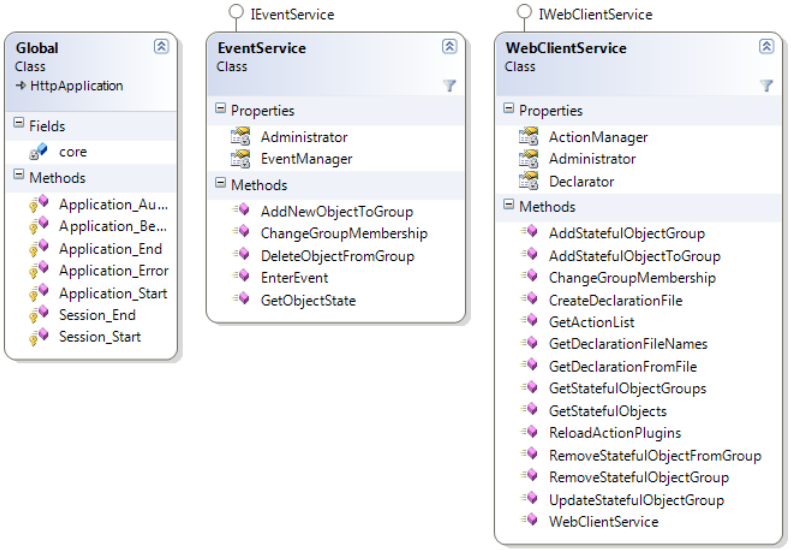


Abbildung (7.25) Klassendiagramm PresenceEngine.Service

Klassenname	Beschreibung
Eventservice	Dieser Service bildet die Schnittstelle zu den Fremdsystemen. Er enthält die Funktionen um der Presence Engine einen Event zur Verarbeitung zu übergeben.
Global	Der Startpunkt der Applikation.
WebClientService	Dies ist die Schnittstelle zum User Interface.

Tabelle (7.10) Package PresenceEngine.Service

PresenceEngine.UI

Dieses Projekt enthält alle für das User Interface benötigten Klassen. Wie das User Interface funktioniert und welchem Zweck die Packages Model, View und ViewModel dienen wird in 7.4.1 *MVVM Model-View-ViewModel Pattern* (siehe Seite 81) beschrieben.

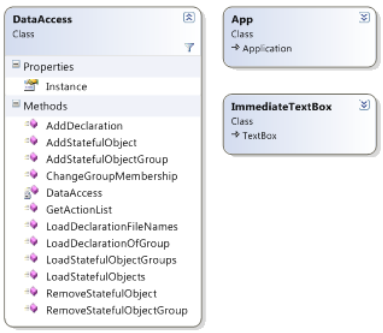


Abbildung (7.26) Klassendiagramm PresenceEngine.UI

Klassenname	Beschreibung
App	In dieser Klasse wird das User Interface gestartet.
DataAccess	Diese Klasse ruft die Daten über den Service auf.
ImmediateTextBox	Hilfsklasse, um Textänderungen ohne Verzögerung zu erfassen.

Tabelle (7.11) Package PresenceEngine.UI

PresenceEngine.UI.Data

Hier werden die lokalen Daten des UIs verwaltet. Die Klassen enthalten jeweils Collections für die diversen Datenlisten.

Abbildung (7.27) Klassendiagramm PresenceEngine.UI.Data

PresenceEngine.UI.Model Hier sind die Models enthalten für das User Interface. Die dienen der Datenaufbereitung für das ViewModel. Die Klassen, welche mit I beginnen sind die jeweils zugehörigen Interfaces.

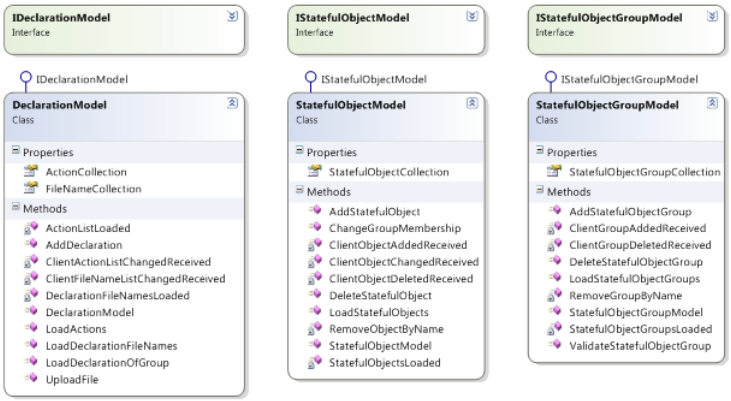


Abbildung (7.28) Klassendiagramm PresenceEngine.UI.Model

PresenceEngine.UI.View Hier sind die Views enthalten für das User Interface. Sie definieren die Darstellung des User Interfaces.

Abbildung (7.29) Klassendiagramm PresenceEngine.UI.View

PresenceEngine.UI.ViewModel Dieses Package enthält die ViewModels für das User Interface.

Abbildung (7.30) Klassendiagramm PresenceEngine.UI.ViewModel

PresenceEngine.Test

Dieses Projekt enthält sämtliche Unit-Tests.

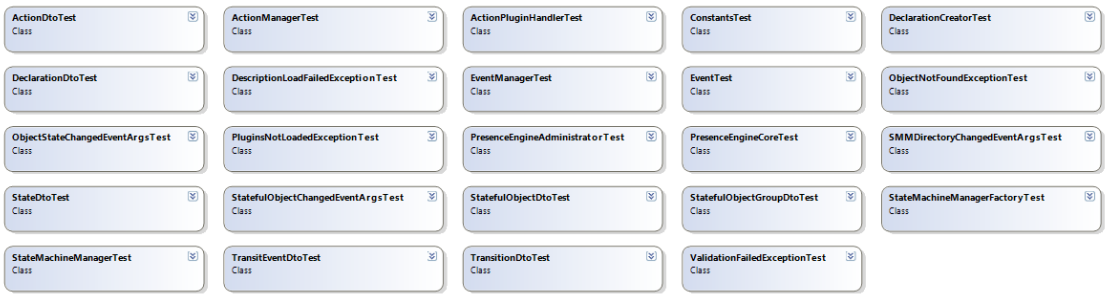


Abbildung (7.31) Klassendiagramm PresenceEngine.Test

7.6 Datenspeicherung

Für die Persistierung der Objekte wird eine SQL Express Datenbank verwendet. Die Objekt Tabelle speichert den Namen des Objektes, dessen Gruppenzugehörigkeit und den aktuellen State.

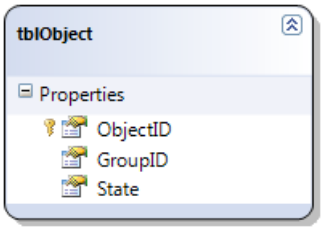


Abbildung (7.32) Datenbankschema

```
CREATE TABLE tblObjects(
        ObjectID nvarchar(50) PRIMARY KEY,
        GroupID nvarchar(50) NOT NULL,
        State nvarchar(50) NOT NULL);
```

Abbildung (7.33) SQL Deklaration der Objekt Tabelle

Per Default ist der SQL Express Server nicht von extern erreichbar. Um dies zu gewährleisten muss das TCP/IP Protokoll im SQL Server Configurationsmanager erlaubt werden. Zudem muss der Port bei der Firewall freigegeben werden. Der Default Port des SQL Express Servers ist TCP 1433 Port.

Die Deklarationen der Gruppen werden in XML Files gespeichert, wodurch die Gruppen nicht in der Datenbank hinterlegt werden müssen. Die Actions werden ebenfalls ausserhalb der Datenbank persistiert, sie sind in dll Files auf dem Server gespeichert.

Aus zeitlichen Gründen konnte die Persistierung nur Teilweise umgesetzt werden. Die Funktionsfähigkeit konnte nicht garantiert werden, da vereinzelt Fehler auftraten und die Zeit nicht ausreichte, um diese Fehlerquellen systematisch zu analysieren. Deshalb wurde die Persistierung nur bis zum Datenbankhandlerinterface umgesetzt. Die Aufrufe innerhalb der Kernapplikation sind realisiert und das Interface verfügt über sämtliche Funktionen.

7.7 Implementierung

7.7.1 Deliverables

PresenceEngine

Wird die Applikation gebuildet resultiert ein Web Deployment Package. Dieses kann auf dem Zielserver deployed werden wie in der *Installationsanleitung* (siehe Seite 143) beschrieben.

PresenceEngine.ActionPlugInContract.dll

Wenn dieses Projekt gebuildet wird resultiert eine Klassenbibliothek (DLL) namens PresenceEngine.ActionPlugInContract.dll, welche das Interface für Action-Plugins enthält. Es kann direkt in anderen Projekten eingebunden werden. Wie in 7.4.3 *Plugin Infrastruktur für Aktionen* (siehe Seite 87) beschrieben muss es in Action-Plugins implementiert werden, damit diese importiert werden können.

8 Testdokumentation

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	09.06.2011	Ricardo Alvarez	System Testfälle erstellt
0.2	10.06.2011	Ricardo Alvarez	System Test ausgeführt
0.3	14.06.2011	Ricardo Alvarez	Zweiter System Test ausgeführt
0.4	14.06.2011	Ricardo Alvarez	Unit Tests

8.1 Unit Test

Der Kern der Applikation ist soweit möglich durch Unit Tests getestet. Dies Umfasst die Projekte PresenceEngine.Common und PresenceEngine.Core, welche fast vollständig durch Unit Tests abgedeckt.

Das User Interface könnte man dank der Implementation mit dem MVVM-Pattern auch durch Unit Tests prüfen, aber diese Tests mussten aus Zeitmangel weggelassen werden. Dasselbe gilt für Tests der Services, wobei diese keine Logik enthalten, sondern nur Funktionen im Kern aufrufen, welche getestet sind. Das die Funktionen aufgerufen werden wird im System Test geprüft. Das Datenbank Projekt ist auch nicht durch Unit Tests abgedeckt, da keine testbaren Funktionen enthalten sind.

8.2 System Test

8.2.1 Testfälle

State Machine Deklaration erstellen

Die Deklaration wird im UI erstellt und gespeichert. Sie soll sowohl States wie auch Transitions enthalten und in diesen werden auch Aktionen hinzugefügt. Das gespeicherte File wird durch erneutes Laden überprüft und die XML Datei wird auch von Hand geöffnet und überprüft.

State Machine Deklaration verwalten

Eine Deklaration wird von der Datei geladen und geändert. Nach dem speichern erfolgt eine Überprüfung wie bei (auf dieser Seite) durchgeführt.

State Machine für Objektgruppe verwalten

1. Eine Deklaration wird geladen und die entsprechende Gruppe wird instanziiert. Die Gruppe erscheint im User Interface. Zur Prüfung werden der Gruppe Objekte hinzugefügt und wieder gelöscht.

2. Eine bestehende Objektgruppe wird wieder gelöscht. Die Aktion ist erfolgreich, wenn die Gruppe im User Interface verschwindet.

Objekt einer Gruppe hinzufügen und Verwalten des Objekts

1. Ein Objekt wird einer bestehenden Gruppe hinzugefügt. Das Objekt erscheint im User Interface bei der Gruppe.

2. Ein erstelltes Objekt wird gelöscht. Das Objekt verschwindet bei der Gruppe im User Interface.

3. Ein erstelltes Objekt wird von einer Gruppe in die andere verschoben. Der Gruppenwechsel kann im User Interface beobachtet werden.

Eventbasierte Statusänderung und Event Schnittstelle

Dieser Use Case kann mit Hilfe eines zusätzlichen Projekts getestet werden. Dieses sendet Events an den Web Service. Mit Hilfe des User Interfaces kann die Verarbeitung der Events beobachtet werden. Folgende Events werden getestet:

1. Ein existierender Event für ein existierendes Objekt. Die Statusänderung sollte beobachtet werden.

2. Ein existierender Event für ein nicht existierendes Objekt. Dieser Event wird ohne Änderung verworfen.

3. Ein nicht existierender Event für ein nicht existierendes Objekt. Dieser Event wird ohne Änderung verworfen.

4. Den Status eines Objekts abrufen. Der Status kann durch Ausgabe im Hilfsprogramm verglichen werden.

5. Ein neues Objekt erstellen. Das Objekt erscheint im User Interface.

6. Ein erstelltes Objekt löschen. Das Objekt verschwindet im User Interface.

7. Ein erstelltes Objekt in eine andere Gruppe verschieben. Das Objekt wechselt im User Interface die Gruppe.

8.2.2 Test Ausführung vom 10.06.2011

State Machine Deklaration erstellen

Die Deklaration kann Erstellt werden. States, Events und Transitions können hinzugefügt werden.

Probleme: Die Validierung der Deklaration funktioniert nicht korrekt.

State Machine Deklaration verwalten

Das Bearbeiten der Deklaration funktioniert. Sämtliche Änderungen wurden übernommen.

Probleme: Wie beim Erstellen der Deklaration funktioniert die Validierung nicht korrekt. Dies ist aber durch dasselbe Problem hervorgerufen.

State Machine für Objektgruppe verwalten

Sämtliche Funktionen beim Verwalten von Objektgruppen haben funktioniert.

Objekt einer Gruppe hinzufügen und Verwalten des Objekts

Sämtliche Funktionen beim Verwalten von Objekten haben funktioniert.

Eventbasierte Statusänderung und Event Schnittstelle

Das Hilfsprogramm kann den Event Service ansprechen und die Events übermitteln.

1. Existierende Events für existierende Objekte werden korrekt verarbeitet.
2. Existierende Events für nicht existierende Objekte werden korrekt verworfen.
3. Nicht existierende Events für nicht existierende Objekte werden korrekt verworfen.
4. Der Status eines Objekts kann korrekt abgerufen werden. Abfragen für nicht bekannte Objekte werden wie vorgesehen verworfen.
5. Objekte können über die Schnittstelle erstellt werden und werden im User Interface angezeigt.

Probleme: Beim Erstellen, Löschen und Verschieben von Objekten sind Probleme mit der Datenbank aufgetreten. Diese müssen erst gelöst werden, bevor die Funktionen erneut getestet werden können.

8.2.3 Test Ausführung vom 14.06.2011

State Machine Deklaration erstellen

Die Deklaration kann Erstellt werden. States, Events und Transitions können hinzugefügt werden. Es sind keine Probleme aufgetreten.

State Machine Deklaration verwalten

Das Bearbeiten der Deklaration funktioniert. Sämtliche Änderungen wurden übernommen. Auch in diesem Test sind keine Probleme aufgetreten.

State Machine für Objektgruppe verwalten

Sämtliche Funktionen beim Verwalten von Objektgruppen haben funktioniert.

Objekt einer Gruppe hinzufügen und Verwalten des Objekts

Sämtliche Funktionen beim Verwalten von Objekten haben funktioniert.

Eventbasierte Statusänderung und Event Schnittstelle

Das Hilfsprogramm kann den Event Service ansprechen und die Events übermitteln.

1. Existierende Events für existierende Objekte werden korrekt verarbeitet.
2. Existierende Events für nicht existierende Objekte werden korrekt verworfen.
3. Nicht existierende Events für nicht existierende Objekte werden korrekt verworfen.
4. Der Status eines Objekts kann korrekt abgerufen werden. Abfragen für nicht bekannte Objekte werden wie vorgesehen verworfen.
5. Objekte können über die Schnittstelle erstellt werden und werden im User Interface angezeigt.

Alle Funktionen der Event Schnittstelle haben funktioniert wie vorgesehen, ohne dass ein Fehler aufgetreten wäre.

9 Persönliche Berichte

9.1 Reto Brühwiler

Projektverlauf

Unser Wunschthema wurde leider einer anderen Gruppe zugesprochen, dadurch mussten wir uns auf ein anderes Thema einigen. Nach einem Meeting mit Herrn Stettler hatten wir uns auf die Arbeit Presence Engine im Spitalumfeld geeinigt. Die Arbeit hat uns sehr viel Freiraum gelassen, fast ein bisschen zu viel. Es fiel uns schwer einen Einstieg in die Arbeit zu finden, da die Arbeit sehr offen Definiert war und wir keinen richtigen Ansatz gefunden hatten. Ich hatte Schwierigkeiten mir das ganze praktisch vorzustellen. Zudem hatten wir zu Beginn einige Kommunikationsschwierigkeiten im Team sowie auch zu den Betreuern.

Dies war unsere erste gemeinsame Arbeit, was die Sache nicht gerade erleichtert. Wir brauchten etwas Zeit um uns aneinander zu gewöhnen, bis die Zusammenarbeit klappte. Durch all diese Faktoren lief die Arbeit ziemlich harzig an. Diese Situation drückte auf die Stimmung und beeinträchtigte leider auch meine Motivation.

In der ersten Phase des Projektes evaluierten wir die Technologien, mit welchen wir das ganze umsetzten. In diesem Projektabschnitt legten wir an regelmässigen Sitzungen fest, welche Funktionalitäten unser Produkt aufweisen soll.

Anschliessend entwickelten wir mögliche Architekturformen und verglichen diese mit Hilfe von Prototypen. Diese interessante Arbeit steigerte meine Motivation. Die Zusammenarbeit im Team funktionierte nun auch. Dies was deutlich am Fortschritt der Arbeit zu sehen und steigerte die Motivation zusätzlich. Nun kamen wir zügig voran und konnten, nach dem wir die Prototypen und unseren Architektur entscheid vorgestellt hatten, mit der Implementierung beginnen. Die restliche Zeit kam mir sehr kurz vor. Was wohl darauf zurück zu führen ist, dass diese Arbeit sehr interessant und spannend war.

Gelerntes

In der Evaluationsphase hatten wir uns für .Net entschieden. Da ich erst wenig Erfahrung im Bereich .Net, C# und Silverlight hatte musst ich mir vieles selber beibringen und in Foren recherchieren. Dadurch konnte ich vieles lernen. Neben dem neu erlernten konnte ich auch einiges verbessern, was ich früher falsch oder zumindest unsauber verwendet hatte. So zum Beispiel das MVVM Pattern in Silverlight.

In diesem Projekt hatte ich zum ersten Mal mit LaTeX gearbeitet. Der Einstieg war ungewohnt und ich war froh, dass ich Ricardo fragen konnte wenn ich nicht weiter wusste.

Fazit

Mit dem Start des Projektes bin ich gar nicht zufrieden, dort hätte einiges viel besser und somit auch zeitsparender ablaufen können.

Abgesehen vom schwierigen Start, ist das Projekt gut verlaufen. Die zu Beginn festgelegten Funktionen konnten praktisch alle erfüllt werden. Es hätte noch einige weitere Ideen gegeben welche ich noch gerne umgesetzt hätte. Diese konnten jedoch aus zeitlichen Gründen nicht mehr realisiert werden.

9.2 Ricardo Alvarez

Projektverlauf

Der Anfang des Projekts war sehr schwierig. Wir hatten Mühe, uns in der offenen Aufgabenstellung zurechtzufinden und einen Ansatz zu finden. Zusätzlich hatten wir im Team Probleme sowohl bei der Kommunikation wie auch bei der Zusammenarbeit. Durch Gespräche in Team konnten wir diese Probleme aber lösen und haben einen Weg gefunden zusammen zuarbeiten. Auch den Ansatz zur Aufgabe haben wir durch Sitzungen mit den Betreuern gefunden. So konnten wir etwas verspätet mit dem Projekt beginnen.

Der erste Teil der Arbeit war eine Technologie Studie, um herauszufinden mit welcher Technologie wir das Projekt überhaupt realisieren wollen. Dies war sehr interessant, da wir sehr viele Ansätze untersucht haben und viel Neues erfahren haben. Allerdings war es auch sehr fordernd, da ich bisher noch nie ohne Einschränkung nach einer passenden Technologie suchen musste. Die Entscheidung ist dann für .Net von Micorsoft ausgefallen, da uns die State Machine sehr überzeugt hat. Dies, obwohl wir beide wenig Erfahrung mit dieser Technologie hatten. Danach mussten wir analysieren, wie die Architektur der Applikation aussehen kann, da sie sehr performant sein muss, um so viele Objekte zu verwalten.

Der zweite Teil der Arbeit bestand dann im programmieren der Presence Engine. Nachdem die Anforderungen spezifiziert waren, konnten wir mit dem Erstellen der Applikation beginnen. Dieser Teil des Projekts war auch sehr spannend und motivierend. Gegen Ende des Projekts wurde dann leider die Zeit etwas knapp, aber wir konnten die Anforderungen fast alle umsetzen.

Gelerntes

Da ich mit dem .Net Framework und C# vorher wenig Erfahrung hatte, gab es sehr viel Neues zu lernen. Besonders das Managed Extensibility Framework für das Erstellen der Plugin Infrastruktur und die WCF Services, welche ich vorher noch nicht benutzt hatte. Auch in LaTeX, das wir für die Dokumentation benutzt haben, konnte ich mein Wissen erweitern.

Fazit

Das Projekt stellt eine interessante und wertvolle Erfahrung dar und ich habe in verschiedenen Aspekten vieles gelernt. Das Programmieren mit C# und .Net hat mir gut gefallen und ich möchte auch zukünftig mein Wissen in diesem Bereich erweitern. Aber auch im zwischenmenschlichen Bereich war es eine gute Erfahrung und hat mir viel gebracht.

Teil III

Projektmanagement

10 Projektorganistion

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	07.03.2011	Ricardo Alvarez	Dokument erstellt

10.1 Methodik

Das Projektmanagement wird gemäss den Richtlinien des agilen Software Entwicklungsprozesses Extreme Programming geführt.

Für die Dokumentation wird eine abgespeckte Variante des Unified Process verwendet.

10.2 Schnittstellen

Name	**Email**	**Funktion**
Beat Stettler	beat.stettler@ins.hsr.ch	Betreuer
Marco Facetti	marco.facetti@ins.hsr.ch	Co-Betreuer
Maurin Egler	maurin.egler@ins.hsr.ch	Co-Betreuer

Tabelle (10.1) Liste der beteiligten Personen

11 Planung

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	07.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	14.04.2011	Ricardo Alvarez	Überarbeitung

11.1 Zeitplanung

Die Backlogs sind unter *Zeitplan* (siehe Seite 157) ersichtlich.

11.2 Meilensteine

Die Tabelle zeigt die geplanten Meilensteine mit den jeweiligen Arbeitsresultaten.

Nr.	**Name**	**Arbeitsresultate**	**Ende**
MS1	Analyse	Analyse der bestehenden Technologie und Anforderungsspezifikation	SW 6
MS2	Prototyp	Prototyp des Lösungsansatzes	SW 10
MS3	Abgabe	Abgabe des Projekts	SW 16

Tabelle (11.1) Meilensteine

11.3 Reviews

Im Rahmen einer wöchentlichen Sitzung mit den Betreuern werden die wichtigen Themen besprochen und Entscheide über das weitere Vorgehen gefällt. Traktanden, Beschlüsse und anstehende Tasks werden im jeweiligen Sitzungsprotokoll festgehalten.

12 Infrastruktur

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	07.03.2011	Ricardo Alvarez	Dokument erstellt

12.1 Hardware

- HSR Arbeitsrechner in den Übungsräumen der HSR
- Private Laptops der Teammitglieder
- Drucker der HSR
- Versionsverwaltungsserver der HSR
- Virtual Private Server (VPS) der HSR als Projektserver

12.2 Software

12.2.1 Dokumentation

- MiKTeX
- TeXnicCenter
- JabRef
- Microsoft Office 2007
- Enterprise Architect

12.2.2 Entwicklungsumgebung

- Microsoft Visual Studio 2010

12.2.3 Infrastuktur-Services

- Hudson

12.2.4 Versionsverwaltung

- Subversion

12.3 Kommunikation

12.3.1 Kommunikationsmittel

- Besprechung
- E-Mail
- Telefon
- Messenger

13 Qualitätssicherung

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	07.03.2011	Ricardo Alvarez	Dokument erstellt
0.2	14.04.2011	Ricardo Alvarez	Überarbeitung

13.1 Qualitätsmassnahmen

13.1.1 Dokumente

Damit das Projekt eine hohe Qualität erreichen kann und sich die Gruppenmitglieder auf deren Inhalt verlassen können, werden die Dokumente stets aktuell gehalten. Zudem werden sämtliche Dokumente von einem weiteren Teammitglied gegengelesen, korrigiert und mit dem Verfasser nachbesprochen.

13.1.2 Sitzungen

Über sämtliche Sitzungen wird Protokoll geführt und Traktanden vorgängig den Beteiligten zugesendet. Das schriftliche Festhalten des Besprochenen hilft weitere Arbeiten zu koordinieren und eventuelle Missverständnisse auszuräumen.

Die Protokolle werden den Betreuern bis spätestens dem darauffolgenden Tag zur Information weitergeleitet.

13.1.3 Codequalität und Codestyle

Codequalität und Stil halten sich an die Microsoft Framework Design Guidelines. Zur Überprüfung des Codes wird NDepend eingesetzt. Allenfalls werden die Guidelines noch um Custom Guidelines erweitert.

Weiter werden Team-intern regelmässig Architektur-Reviews durchgeführt und diese mit Analyse-Tools wie z.B. NDepend auch analysiert.

13.2 Risiko Analyse

Die Risiko Analyse befindet sich im Anhang unter F *Risiko Analyse* (siehe Seite 167) .

13.3 Tests

13.3.1 Unit-Tests

Sämtlicher Code soll wo möglich mit Unit-Tests geprüft werden. Die Unit-Tests werden automatisch durchgeführt, wenn der Code eingecheckt wird.

13.3.2 System Tests

Mit dem System Test wird überprüft, ob die Anforderung richtig umgesetzt wurden und das Programm keine Fehler in den Abläufen enthält.

Literaturverzeichnis

[AA10] Albahari, Joseph und Albahari, Ben: *C# 4.0 in a Nutshell*, O'Reilly Media (2010)

[Act] Activiti BPM Platform, http://www.activiti.org/; letzter Zugriff: 14.06.2011

[bbv] bbv.common: bbv.common .Net component library, http://code.google.com/p/bbvcommon; letzter Zugriff: 14.06.2011

[Dol] Dolinger, Jason: Jason Dolinger on Model-View-ViewModel, http://blog.lab49.com/archives/2650; letzter Zugriff: 14.06.2011

[JBo] JBoss: jBPM, http://www.jboss.org/jbpm; letzter Zugriff: 14.06.2011

[MEF] MEF community site, http://mef.codeplex.com/; letzter Zugriff: 14.06.2011

[Mic10a] Microsoft: MSDN Library: Managed Extensibility Framework Overview (2010), http://msdn.microsoft.com/en-us/library/dd460648.aspx; letzter Zugriff: 14.06.2011

[Mic10b] Microsoft: MSDN WCF: Getting Started Tutorial (2010), http://msdn.microsoft.com/en-us/library/ms734712.aspx; letzter Zugriff: 14.06.2011

[Mic10c] Microsoft: MSDN Windows Workflow Foundation (2010), http://msdn.microsoft.com/en-us/library/ms735967(v=vs.90).aspx; letzter Zugriff: 14.06.2011

[Pij08] Pijanowski, Keith: MSDN WF How-To: Building State Machines with Windows Workflow Foundation (2008), http://msdn.microsoft.com/en-us/magazine/cc163281.aspx; letzter Zugriff: 14.06.2011

[Smi] Smith, Josh: WPF-Anwendungen mit dem Model-View-ViewModel-Entwurfsmuster, http://msdn.microsoft.com/de-de/magazine/dd419663.aspx; letzter Zugriff: 14.06.2011

[YAW] YAWL: Yet Another Workflow Language, http://www.yawlfoundation.org/; letzter Zugriff: 14.06.2011

Glossar

.Net	Ein von Microsoft entwickeltes Framework bestehend aus einer Laufzeitumgebung, Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen
Activiti	Java basierte Open Source Business Prozess Management Plattform
bbv.common	.Net component library der Firma bbv Software Services AG
BMNN	Beschreibungssprache
BPEL	Beschreibungssprache
BPM	Business Process Management - Dient zum Abbilden von Geschäftsprozessen
CRUD	Create Read Update Delete
DLL	Dynamic Link Library - Dynamische Verbindungsbibliothek
GUI	Graphical User Interface
HSR	Hochschule für Technik Rapperswil
IP	Internet Protocol
LGPL	Lesser General Public License -Open Source Lizenz
MAF	Managed Add-In Framework - Bestandteil von .Net
MEF	Managed Extensibility Framework- Bibliothek, ermöglicht Erweiterungen zur Laufzeit laden
MVC	Model View Controller - Architektur Pattern
MVVM	Model-View-ViewModel Pattern
OMG	Object Management Group
SQL	Structured Query Language - Datenbanksprache
TCP	Transmission Control Protocol

UC	Use Case
UI	User Interface
WCF	Windows Communication Foundation -Dienstorientierte Kommunikationsschnittstelle Bestandteil von .Net
WF	Windows Workflow Foundation
WPF	Windows Presentation Foundation - Grafisches Framework und Bestandteil des .Net Frameworks
WSDL	Extensible Application Markup Language - Spezielle Form von XML für GUI und WF Deklarationen
XAML	Extensible Application Markup Language - Spezielle Form von XML für GUI und WF Deklarationen
XML	Extensible Markup Language Auszeichnungssprache zur Darstellung hierarchischer Strukturen
YAWL	Beschreibungssprache

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Aufgabenstellung

Der Spitalalltag besteht für die Angestellten zu einem beträchtlichen Anteil aus Suchen: Suchen nach freien Geräten, Suchen nach Ärzten und auch Suchen nach Patienten. Als Abhilfe setzt man zunehmend Lokalisierungstechnologien ein. Je nach Objekt wird WLAN-Lokalisierung, RFID oder GPS basierte Lokalisierungen verwendet. Eine Bachelorarbeit beschäftigt sich bereits mit dem Thema, wie Informationen aus verschiedenen Lokalisierungssensoren zu einer ïntelligentenÖrtsangabe verdichtet werden. Zudem kann die Distanz zwischen Objekten berechnet werden.
Allerdings hilft es nichts, ein Gerät zu lokalisieren, das besetzt ist. Neben dem Standort ist also auch der SZustand"(neudeutsch: "Presence") des Objektes von Interesse. Zustände von Geräten könnten beinhalten: Frei, Besetzt, Defekt, Funktionsumfang, Wartungsbedarf usw. Zustände von Personen könnten neben dem Standort sein: Behandelnd, in der Pause, am Operieren, Frei resp. Abwesend, Anwesend aber nicht zuständig usw. Bei Menschen sind zusätzlich die Fähigkeiten wichtig. So nützt es wenig, einen Hautarzt zu alarmieren, wenn ein Patient eine Herzschwäche erleidet.

Ziel: In diesem Projekt soll eine State Machine programmiert werden, welche flexibel genug ist, um die verschiedenen Zustände von Objekten oder Personen zu verwalten. Beim Übergang von einem Zustand zu einem Anderen, müssen mehrere Aktionen (Fremdsysteme über den neuen Zustand informieren, eine Ampel auf Rot setzen usw.) ausgeführt werden können. Dazu stellen wir uns eine klar definierte Plugin-Infrastruktur vor, welche es uns erlaubt, die State Machine beliebig zu erweitern.

Anhang B

Einrichtungsanleitung

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	10.05.2011	Ricardo Alvarez	Dokument erstellt

B.1 Einführung

B.1.1 Zweck

In diesem Dokument wird beschrieben, wie die Entwicklungsumgebung für dieses Projekt eingerichtet werden kann.

B.2 Dokumentationswerkzeuge

Da wir die LaTeX Vorlage für diese Dokumentation von Mischa Trecco erhalten haben, ist die Einrichtungsanleitung weitgehend der Dokumentation seiner Bachelor-Arbeit entnommen.

B.2.1 MiKTeX

Dies ist die benutzte Tex Implementation. Sie kompiliert die LaTeX Dokumente ins PDF Format:

1. MiKTeX von der Webseite miktex.org herunterladen.
2. Installation startet und den Installationsanweisungen folgen.
3. Während der Installation die Option „install missing packages on-the-fly“ auf Yes setzen.

B.2.2 TeXnicCenter

Dies ist der verwendete Editor um die LaTeXDateien zu bearbeiten.

1. TeXnicCenter von der Webseite texniccenter.org herunterladen.
2. Installation startet und den Installationsanweisungen folgen.
3. TeXnicCenter starten und Tool Tip schliessen.
4. Im Konfigurationsassistenten auf Next klicken. Den Pfad zum MiXTeX bin Verzeichnis angeben, normalerweise C:
   Program Files
   MiKTeX 2.9
   miktex
   bin und zweimal auf Next klicken.
5. Auf Next und dann auf Finish drücken.
6. Im TeXnicCenter Menü Ausgabe => Ausgabeprofile definieren. . . auswählen.
7. Auf der linken Seite LaTeX => PDF auswählen.
8. Rechts bei Pfad des MakeIndex-Compilers: makeindex.exe ersetzen durch makeglossaries.exe.

B.2.3 Strawberry Perl

Wird von diversen Packages der LaTeX Umgebung benötigt.

1. Von strawberryperl.com herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

B.2.4 JabRef

Tool für die Verwaltung des Literaturverzeichnisses mit LaTeX.

1. Von jabref.sourceforge.net herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

B.2.5 Tortoise SVN

Der Client für den Versionsverwaltungs-Dienst Subversion.

1. Von tortoisesvn.net herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

B.2.6 Enterprise Architect

Tool um UML-Diagramme zu erstellen.

1. Von sparxsystem.com herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

B.3 Entwicklerwerkzeuge

B.3.1 Microsoft Visual Studio 2010 Ultimate

Entwicklungsumgebung für die Entwicklung von Applikationen mit dem Microsoft .NET Framework.

1. Produkt erwerben.
2. Installation startet und den Installationsanweisungen folgen.

B.3.2 Microsoft Silverlight SDK

Erweiterung für das Visual Studio 2010, um Silverlight Applikationen zu entwickeln.

1. Den Microsoft Web Platform Installer von www.silverlight.net/getstarted herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

B.3.3 NDepend

Visualisierungstool für Software. Anhand von Metriken kann die Codequalität überprüft und verbessert werden.

1. Von ndepend.com herunterladen.
2. Installation startet und den Installationsanweisungen folgen.

Anhang C

Installationsanleitung

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	09.06.2011	Reto Brühwiler	Dokument erstellt

C.1 Inbetriebnahme der Presence Engine

In diesem Dokument wird beschrieben, wie das Web Deployment Package der Presence Engine auf dem Zielserver eingerichtet werden kann.
Folgenden Komponenten sind im Deployment Package enthalten:

PresenceEngine.Service.deploy.cmd Ist die ausführbare Datei, welche die Applikation auf dem IIS Server einrichtet.

PresenceEngine.Service.SetParameters.xml Dies ist die Konfigurationsdatei für das Hosting der Applikation auf dem IIS.

PresenceEngine.Service.SourceManifest.xml Diese Konfigurationsdatei enthält die Pfade der Ressourcen, welche vom Server gehosted werden sollen.

PackageTmp(Ordner) Der Ordner beinhaltet alle vom IIS Server für das Hosting benötigten Dateien.

Um die Applikation zu hosten wird neben dem IIS Server das Web Deploy Tool[1] von Microsoft benötigt. Nach der Installation des Tool müssen die Pfade in der Datei SourceManifest angepasst werde. Bei Bedarf kann der Name der Applikation auf dem IIS in der Datei SetParameter angepasst werden.
Wenn die Einstellungen in den Konfigurationsdateien gemacht wurden, kann die ausführbare Datei in der Command Line des Web Deploy Tools aufgerufen werden. Mit dem /T Parameter kann der Deploy simuliert werden oder direkt mit dem Parameter /Y deployed werden. In beiden fällen wird eine Textdatei erzeugt, welche weitere Informationen enthält. Sollte beim ausführen der cmd Datei ein Problem auftreten, wird die Fehlermeldung sowie weitere Hinweise und mögliche Lösungen in diese Datei geschrieben.

1 Das Web Deploy Tool kann unter folgender Adresse heruntergeladen werden: http://technet.microsoft.com/en-us/library/dd569059%28WS.10%29.aspx

Anhang D

Benutzerhandbuch

Dokumenthistory

Rev.	Datum	Wer	Änderung
0.1	13.06.2011	Reto Brühwiler	Dokument erstellt

D.1 Objekte verwalten

Neues Objekt hinzufügen

Wählen Sie die Gruppe in zu welchem das Objekt gehören soll aus und klicken Sie auf neues Objekt erstellen.

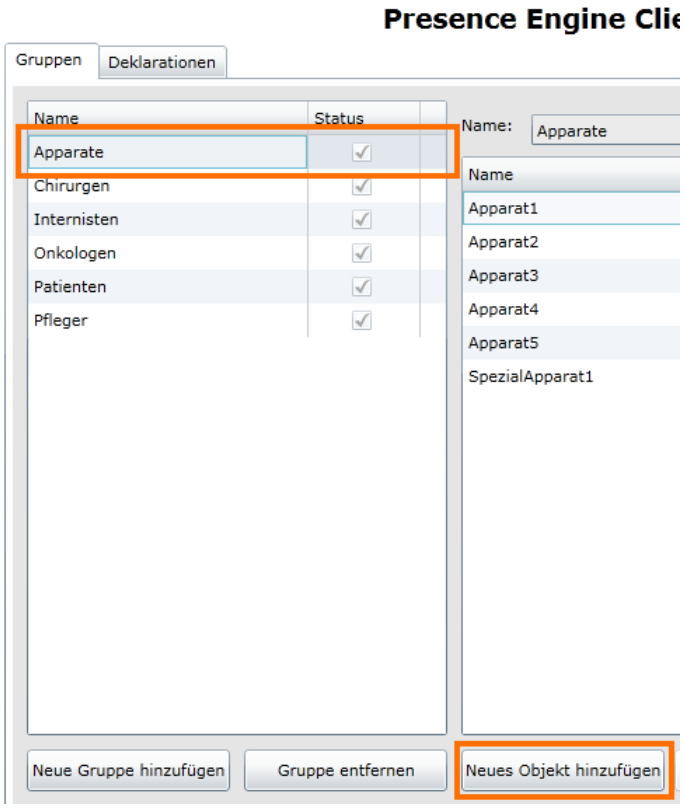


Abbildung (D.1) Neues Objekt erstellen

Es öffnet sich ein Fenster in welchem Sie den Namen des neuen Objektes erfassen können. Bestätigen Sie Ihre Eingabe mit dem Ok Button

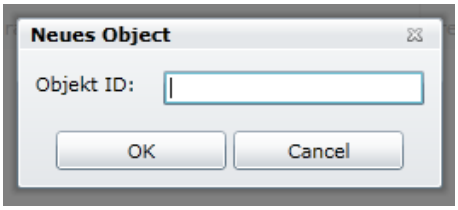


Abbildung (D.2) Neues Objekt erstellen Dialog

Objekt löschen

Wählen Sie das zu löschende Objekt aus welche Sie löschen wollen und klicken Sie auf Objekt löschen.

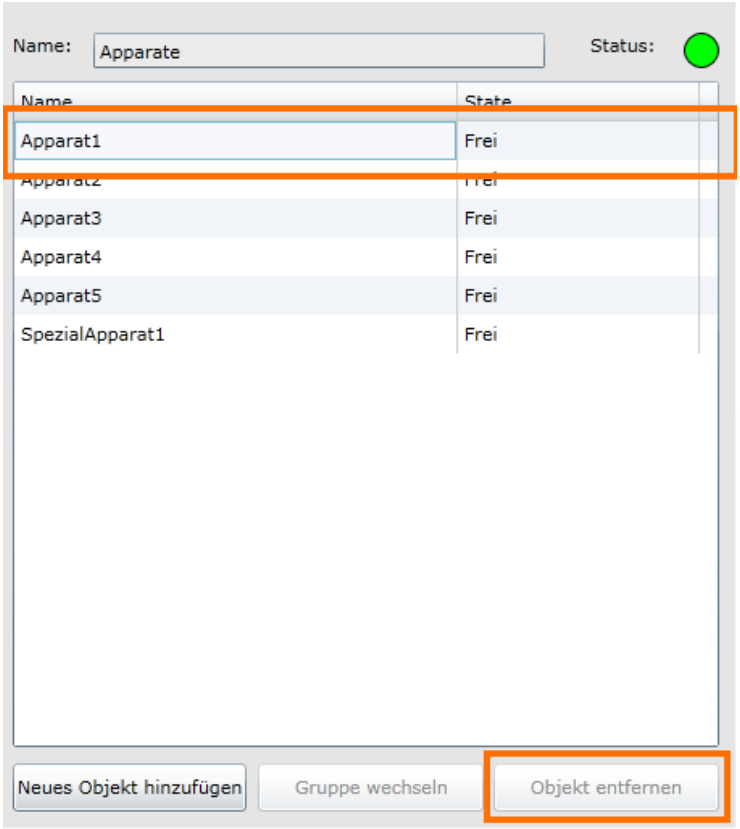


Abbildung (D.3) Objekt entfernen

Objekt in andere Gruppe verschieben

Wählen Sie das zu verschiebende Objekt aus und klicken Sie auf Objekt verschieben.

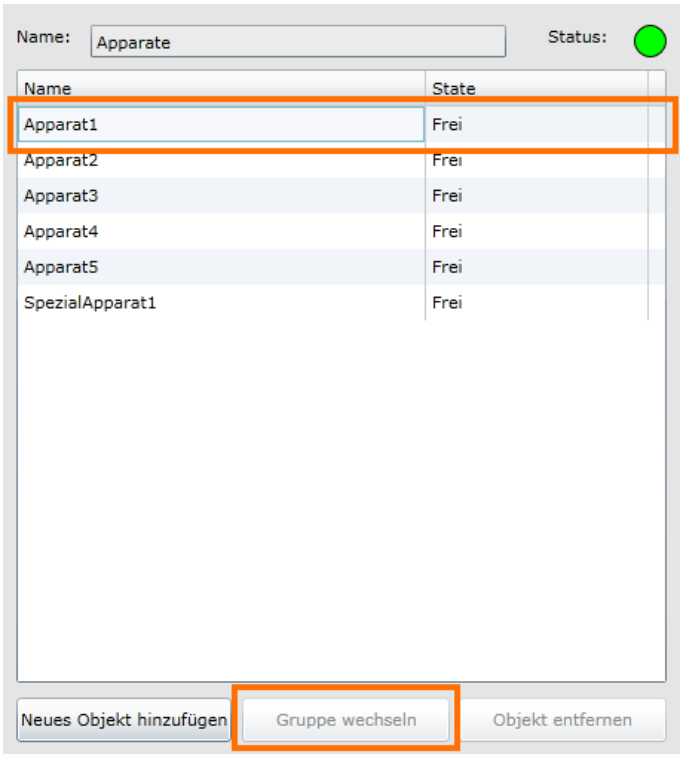


Abbildung (D.4) Objekt in andere Gruppe verschieben

Es öffnet sich ein Dialog in welche Sie die neue Gruppe auswählen können. Es werden nur laufende Gruppen angezeigt. Wählen Sie die gewünschte Gruppe und bestätigen Sie die Eingabe mit dem OK Button.

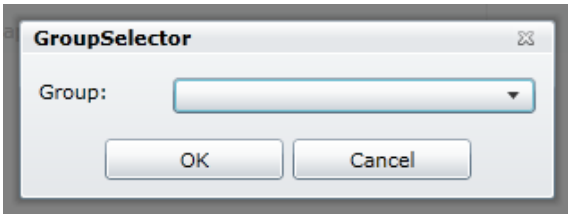


Abbildung (D.5) Gruppen wechseln Dialog

D.2 Gruppe verwalten

Neue Gruppe hinzufügen

Klicken Sie auf neue Gruppe hinzufügen.

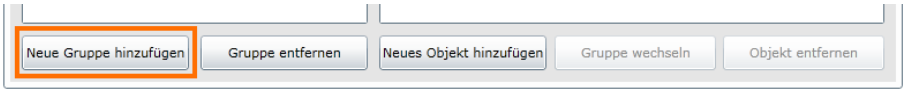


Abbildung (D.6) Neue Gruppe hinzufügen

Es öffnet sich ein Dialog in welchem Sie die gewünschte Gruppe auswählen können. Es werden alle Deklarationen angezeigt welche noch nicht gestartet sind.

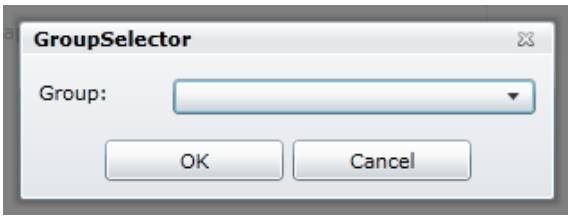


Abbildung (D.7) Neue Gruppe auswählen Dialog

Gruppe löschen

Wählen Sie die zu löschenden Gruppe aus und klicken Sie auf Gruppe

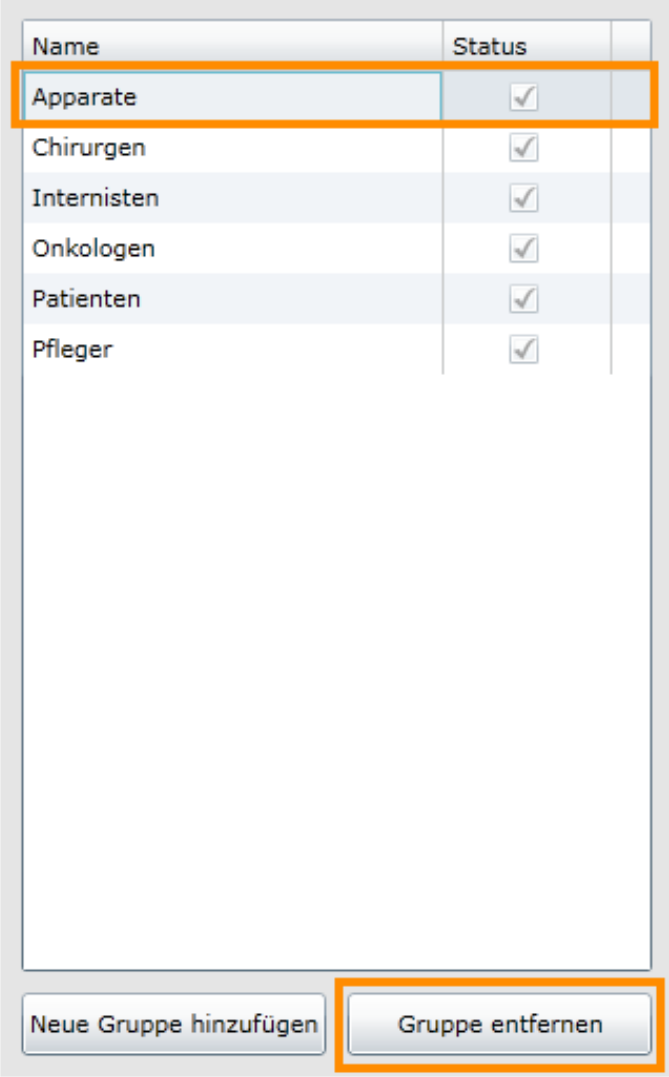


Abbildung (D.8) Gruppe entfernen

D.3 Deklaration verwalten

Wechseln Sie in den Deklaration Tab.

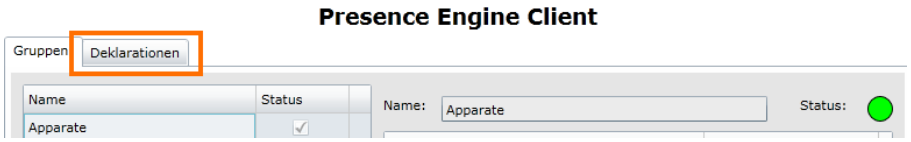


Abbildung (D.9) Zur Deklarationsverwaltung wechseln

Klicken Sie auf neue Deklaration erstellen um eine neue Deklaration zu erstellen. Um eine bestehende Deklaration anzupassen wählen Sie die anzupassende Deklaration aus und klicken Sie auf Deklaration bearbeiten. Der Editor wird geöffnet und Sie können die Deklaration erfassen oder anpassen.

State verwalten

Wechseln Sie im Editor in den Tab States. State hinzuzufügen: Geben Sie den Namen in das Feld (B) ein und klicken Sie auf State hinzufügen (C). Der Button kann nur benutzt werden, wenn der State Name gültig ist. Der State ist ungültig, wenn der Name leer ist oder bereits ein State oder Event mit demselben Namen in dieser Deklaration existiert. State entfernen: Wählen Sie den zu entfernenden State aus (A) und klicken Sie auf State entfernen (D).

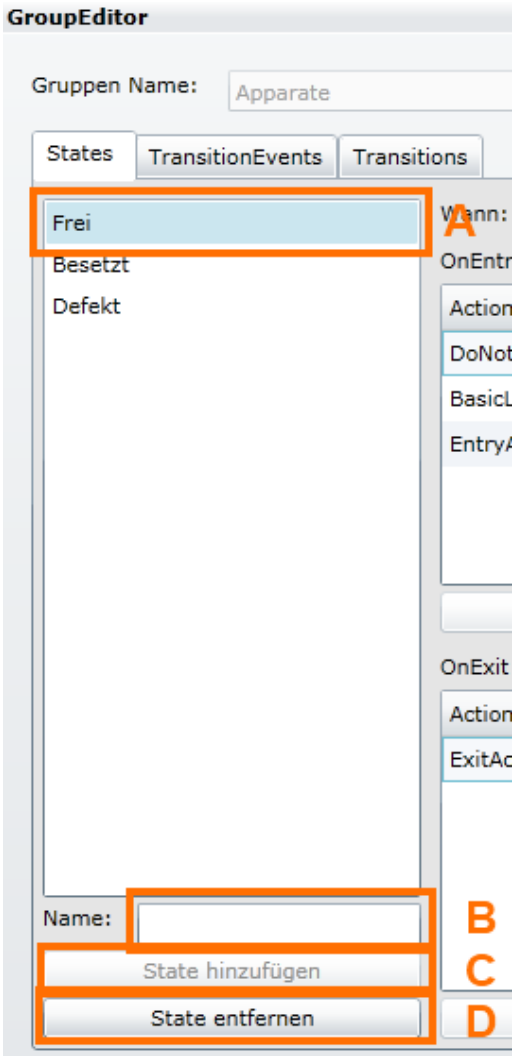


Abbildung (D.10) States der Deklaration verwalten

Event verwalten

Event hinzufügen: Geben Sie den Namen des Events ein und wann dieser verfällt(B). Klicken Sie auf Event hinzufügen(C). Dies ist nur möglich wenn der Event gültig ist. Der Event ist ungültig, wenn der Name leer ist oder bereits ein Event oder State mit demselben Namen in dieser Deklaration existiert.
Event entfernen: Wählen Sie den zu entfernenden Event aus (A) und klicken Sie auf Event entfernen (D).

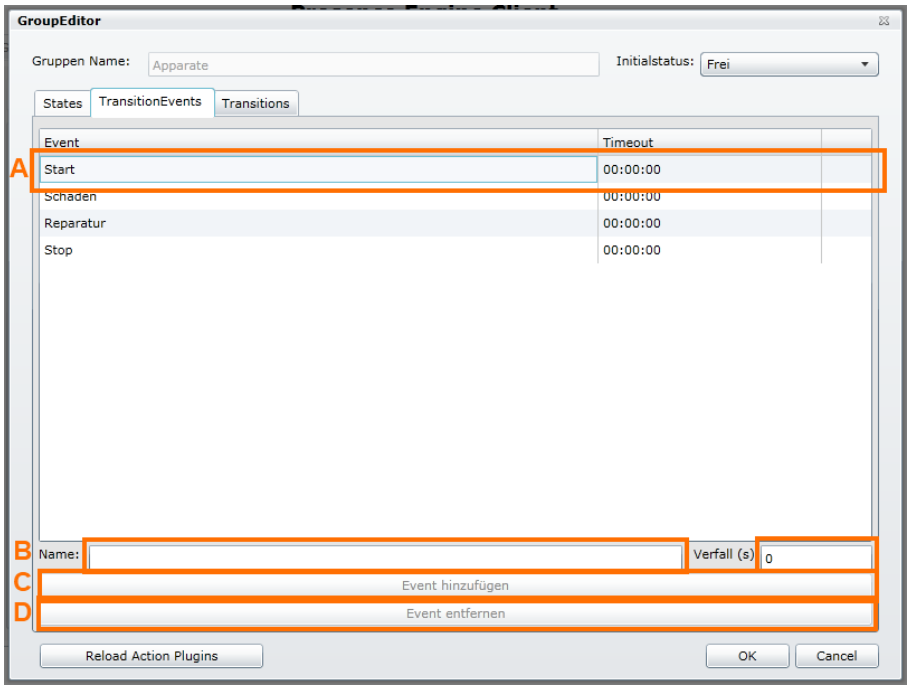


Abbildung (D.11) Events der Deklaration verwalten

Transition verwalten

Wechseln Sie in den Transition Tab.
Transition hinzufügen: Wählen Sie den von State, den nach State und den Event, der die Transition auslöst aus (B) und klicken Sie auf Transition hinzufügen (C). Dies ist nur möglich wenn die Transition gültig ist. Die Transition ist ungültig, wenn der die Kombination Von State und Event bereits existiert.
Tranistion entfernen: Wählen Sie die zu entfernende Transition aus (A) und klicken Sie auf Transition entfernen (D).

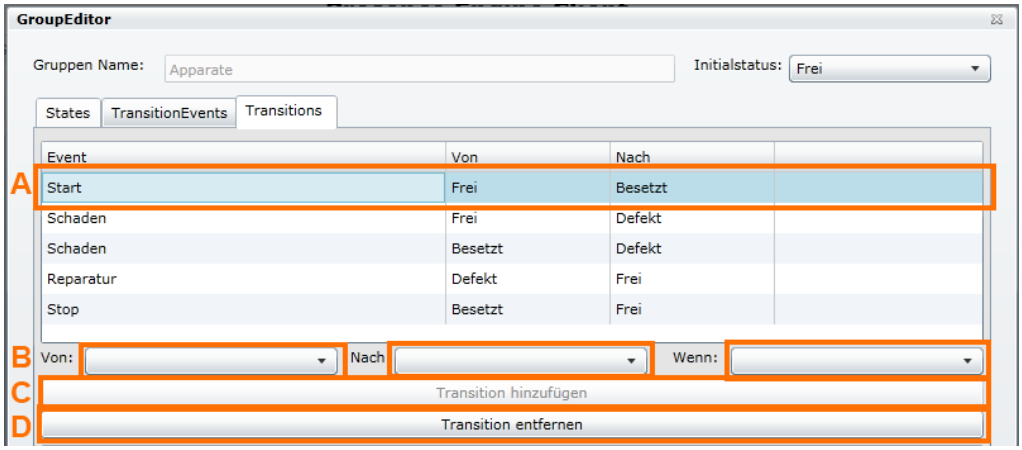


Abbildung (D.12) Transitions der Deklaration verwalten

State Action verwalten

State Actions hinzufügen: Wechseln Sie in den State Tab. Wählen Sie den State (A), wann die Action ausgeführt werden soll sowie die auszuführende Action aus (B). Klicken Sie anschliessend auf Action hinzufügen.
Action entfernen: Wählen Sie die zu entfernende Action aus(C) und klicken Sie auf Action entfernen (D).

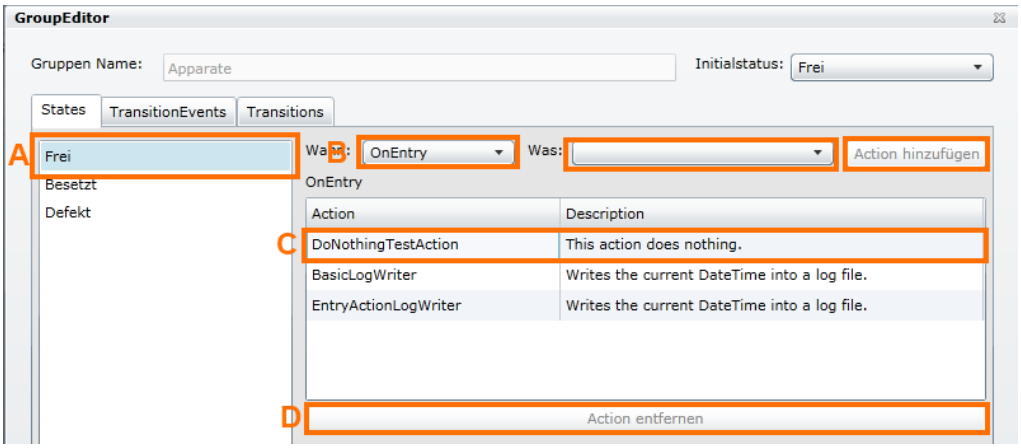


Abbildung (D.13) Aktionen in den States verwalten

Transition Action verwalten

Transition Action hinzufügen: Wechseln Sie in den Transition Tab. Wählen Sie die gewünschte Transition (A) sowie die auszuführende Action (C) aus und klicken Sie auf Aktion hinzufügen (D).
Action entfernen: Wählen Sie die zu entfernende Action (B) aus und klicken Sie auf Action entfernen (E).

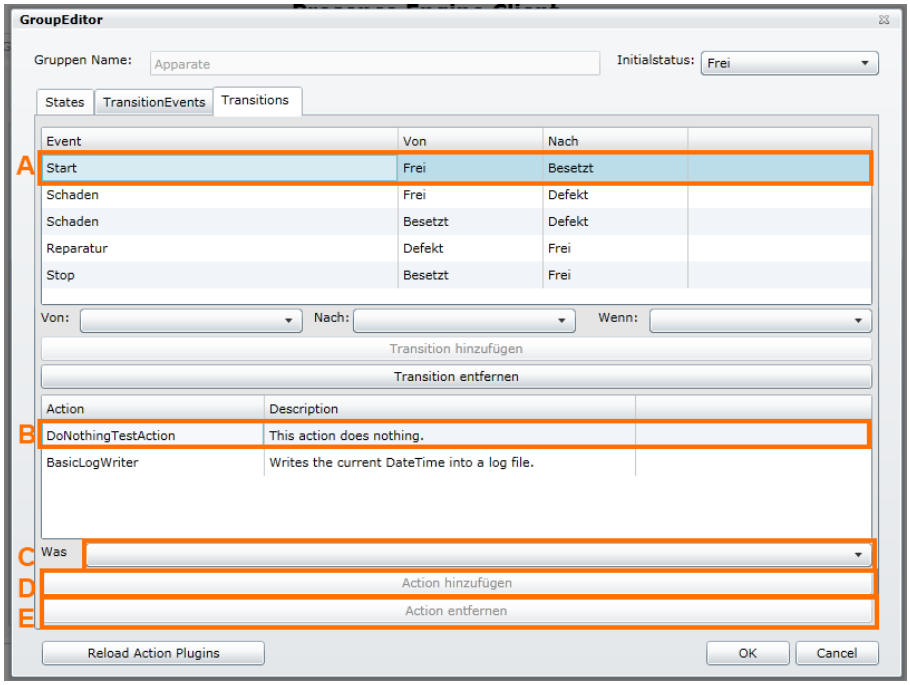


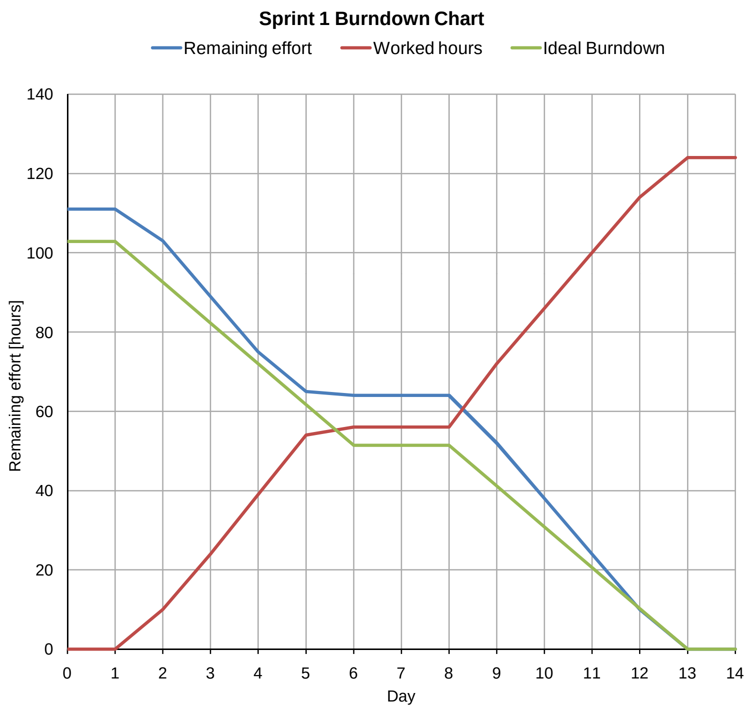
Abbildung (D.14) Aktionen in den Transitions verwalten

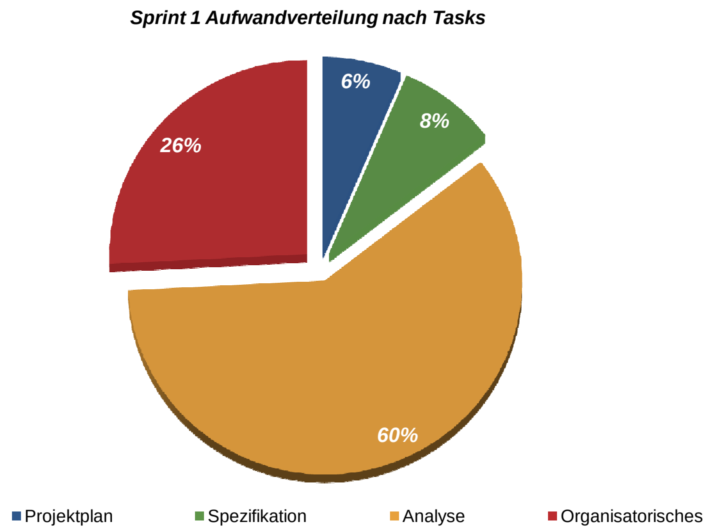
Anhang E

Zeitplan

Sprint 1 Backlog

Nr.	Priority	Status		Task	Verantwortlich	Hours worked	0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So
							Estimated hours remaining per day | Hours worked on task per day																												
1	Medium		**Projektplan**	Dokument erstellen	Ricardo Alvarez	6	5	5		5		5	3	2	3																				
2	Medium			Risikomanagement	Ricardo Alvarez	2	2	2		2		2	2																						
						0																													
3	Medium		**Spezifikation**	Spezifikation erarbeiten	Team	10	10	10		10		10		10		10		10		10		10		10		10		10	6	4	4				
						0																													
4	Medium		**Analyse**	Technologie Evaluation	Team	74	70	70	8	62	6	56		56	10	50		50		50		50	14	36	14	22	14	8	8						
						0																													
5	Medium		**Organisatorisches**	Infrastruktur erstellen	Reto Brühwiler	9	8	8		8	4	4	3	1		1	2																		
6	Medium			Dokumentvorlagen erstellen	Ricardo Alvarez	8	8	8		8	4	4	2	2		2		2		2		2		2		2		2		2	2				
7	Medium			Entscheid Entwicklungsprozess	Reto Brühwiler	4	4	4		4		4	4																						
8	Medium			Meeting	Team	11	4	4	2	4		4	1	4	2	2		2		2		2	2	4		4		4		4	4				
					Estimated work remaining:		**111**	**111**		**103**		**89**		**75**		**65**		**64**		**64**		**64**		**52**		**38**		**24**		**10**					
					Remaining working hours:		**103**	**103**		**93**		**82**		**72**		**62**		**51**		**51**		**51**		**41**		**31**		**21**		**10**		**0**		**0**	
					Hours worked per Day:				**10**		**14**		**15**		**15**		**2**						**16**		**14**		**14**		**14**		**10**				
Sprint finished					**Hours worked total:**	**124**		**0**	**10**	**10**	**24**	**24**	**39**	**39**	**54**	**54**	**56**	**56**	**56**	**56**	**56**	**56**	**72**	**72**	**86**	**86**	**100**	**100**	**114**	**114**	**124**	**124**	**124**	**124**	**124**

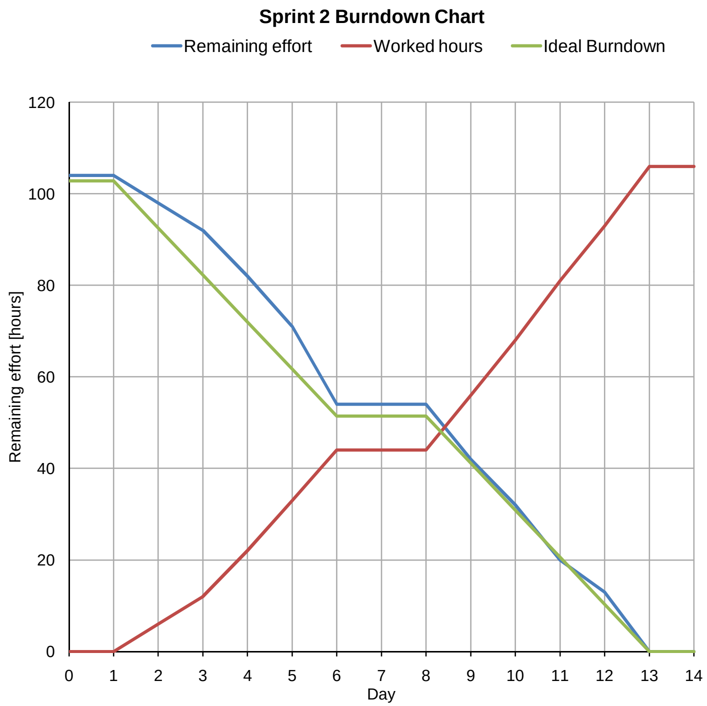


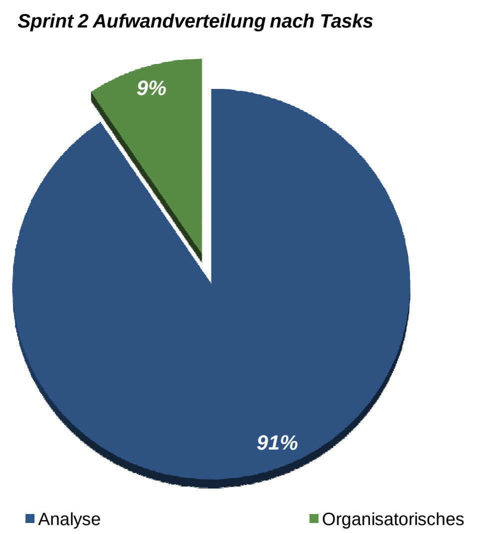


Sprint 2 Backlog

Estimated hours remaining per day | Hours worked on task per day

Nr.	Priority	Status		Task	Verantwortlich	Hours worked	0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So
1	Medium		**Analyse**	Anforderungs Analyse	Team	38	40	40		40		40	5	35	5	30	4	24		24		24	7	17	5	12	5	7	5	2	2				
2	Medium			Architektur Analyse	Team	36	40	40	6	34	6	28		28	3	25	2	19		19		19	3	16	4	12	4	8	5	3	3				
3	Medium			Domain Model	Team	22	20	20		20		20	5	15	3	12	3	9		9		9	2	7	2	6	4	3		3	3				
						0																													
4	Medium		**Organisatorisches**	Infrastruktur	Ricardo Alvarez	5																								5	5				
5	Medium			Meeting	Team	5	4	4		4		4		4		4	2	2		2		2		2	1	2		2	2						
						0																													
						0																													
						0																													
						0																													
						0																													
					Estimated work remaining:		**104**	**104**		**98**		**92**		**82**		**71**		**54**		**54**		**54**		**42**		**32**		**20**		**13**					
					Remaining working hours:		**103**	**103**		**93**		**82**		**72**		**62**		**51**		**51**		**51**		**41**		**31**		**21**		**10**		**0**		**0**	
					Hours worked per Day:				**6**		**6**		**10**		**11**		**11**						**12**		**12**		**13**		**12**		**13**				
Sprint finished					**Hours worked total:**	**106**		**0**	**6**	**6**	**12**	**12**	**22**	**22**	**33**	**33**	**44**	**44**	**44**	**44**	**44**	**44**	**56**	**56**	**68**	**68**	**81**	**81**	**93**	**93**	**106**	**106**	**106**	**106**	**106**

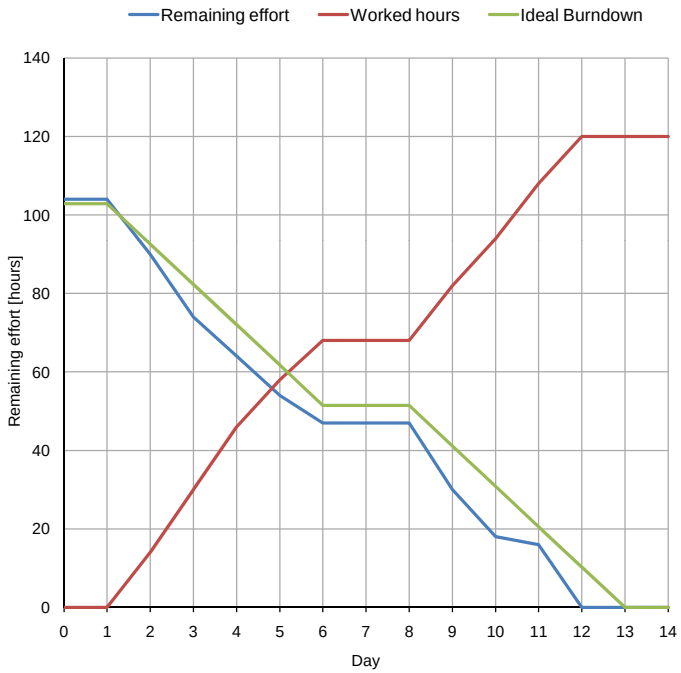




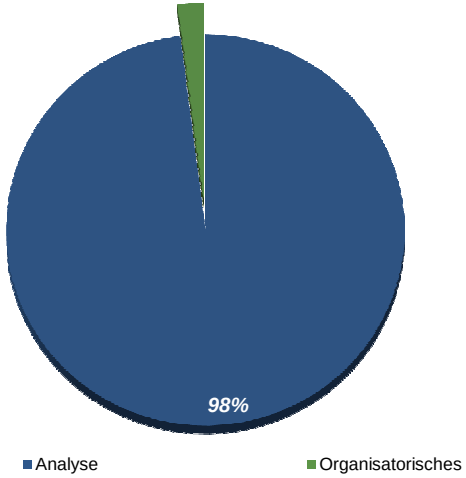
Sprint 3 Backlog

Nr.	Priority	Status		Task	Verantwortlich	Hours worked	Estimated hours remaining per day | Hours worked on task per day																																			
							0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So									
1	Medium		**Analyse**	Technologie Evaluation	Team	10	10	10	6	4	4																																	
2	Medium			Architektur Prototyp	Team	86	70	70	8	62	12	50	16	40	12	30	8	25		25		25	10	12	12			6	8													
3	Medium			Architektur Prototyp Tests	Team	22	20	20		20		20		20		20		20		20		20	4	16		16	14	8	4													
						0																																				
						0																																				
4	Medium		**Organisatorisches**	Meeting	Team	2	4	4		4		4		4		4	2	2		2		2		2		2		2														
						0																																				
						0																																				
						0																																				
						0																																				
						0																																				
					Estimated work remaining:		**104**	**104**		**90**		**74**		**64**		**54**		**47**		**47**		**47**		**30**		**18**		**16**														
					Remaining working hours:		**103**	**103**		**93**		**82**		**72**		**62**		**51**		**51**		**51**		**41**		**31**		**21**		**10**		**0**		**0**								
					Hours worked per Day:				**14**		**16**		**16**		**12**		**10**						**14**		**12**		**14**		**12**														
Sprint finished					**Hours worked total:**	**120**		**0**	**14**	**14**	**30**	**30**	**46**	**46**	**58**	**58**	**68**	**68**	**68**	**68**	**68**	**68**	**82**	**82**	**94**	**94**	**108**	**108**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**	**120**

Sprint 3 Burndown Chart



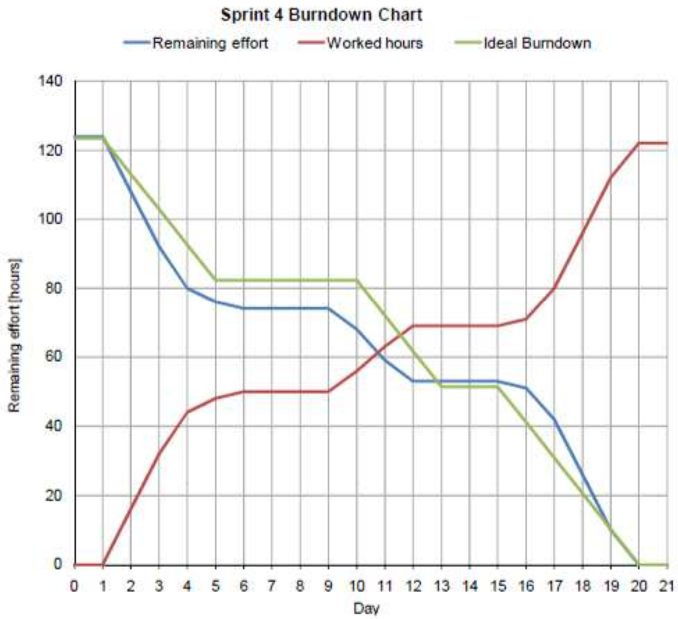
Sprint 3 Aufwandverteilung nach Tasks

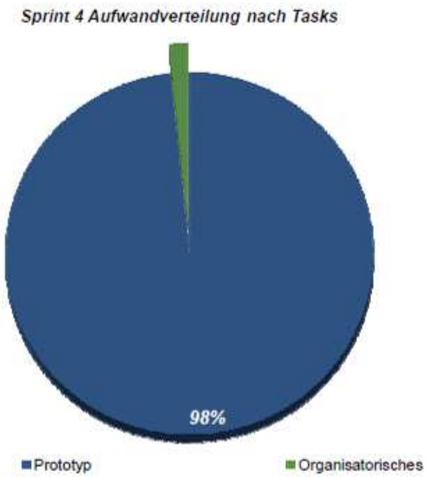


Sprint 4 Backlog

Estimated hours remaining per day | Hours worked on task per day

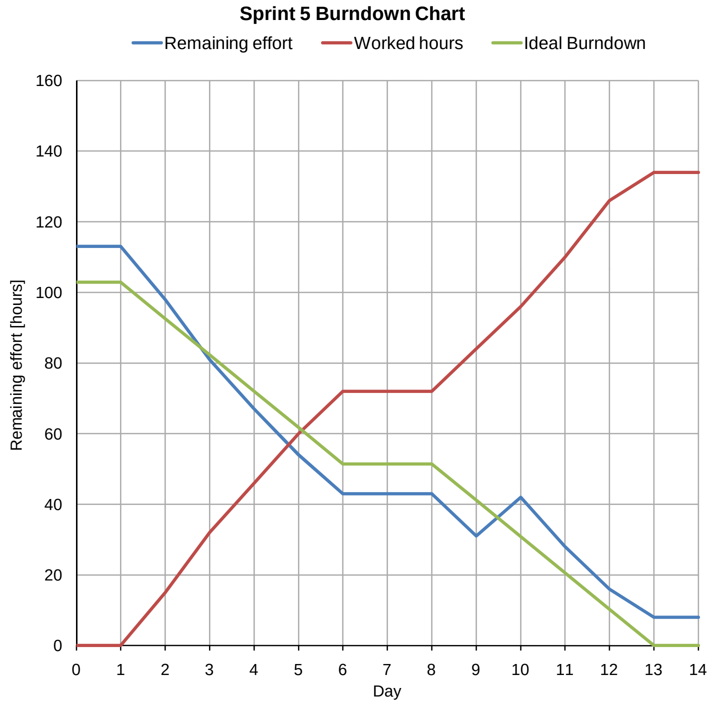
Nr.	Priority	Status		Task	Verantwortlich	Hours worked	0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So	15	Mo	16	Di	17	Mi	18	Do	19	Fr	20	Sa	21	So
1	High		Prototyp	Presence Engine	Ricardo Alvarez	60	60	60	8	52	8	44	6	38	4	34		34		34		34		34	2	32	4	28	4	24		24		24		24		24	4	20	8	12	8	4	4				
2	Medium			User Interface	Reto Brühwiler	60	60	60	8	52	8	44	6	38		38	2	36		36		36		36	4	32	3	29	2	27		27		27		27	2	25	5	20	8	12	8	4	4				
						0																																												
						0																																												
						0																																												
3	Medium		Organisatorisches	Meeting	Team	2	4	4		4		4		4		4		4		4		4		4		4		2		2		2		2		2		2		2		2		2	2				
						0																																												
						0																																												
						0																																												
						0																																												
						0																																												
					Estimated work remaining:		124	124		108		92		80		76		74		74		74		74		68		59		53		53		53		53		51		42		26		10					
					Remaining working hours:		123	123		113		103		92.6		82.3		82.3		82.3		82.3		82.3		82.3		72		61.7		51.4		51.4		51.4		41.1		30.9		20.6		10.3		0		0	
					Hours worked per Day:				16		16		12		4		2								6		7		6								2		9		16		16		10				
Sprint finished					Hours worked total:	122		0	16	16	32	32	44	44	48	48	50	50	50	50	50	50	50	50	56	56	63	63	69	69	69	69	69	69	69	69	71	71	80	80	96	96	112	112	122	122	122	122	122

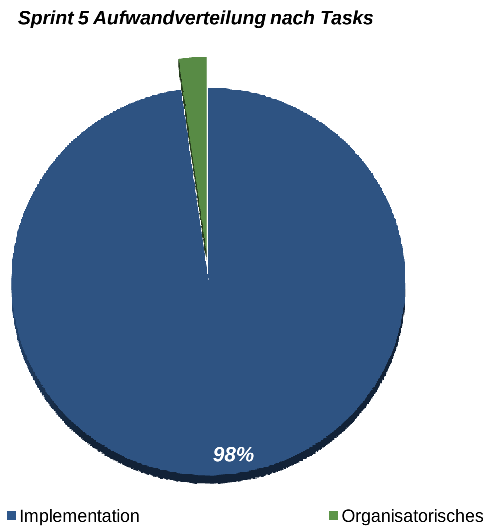




Sprint 5 Backlog

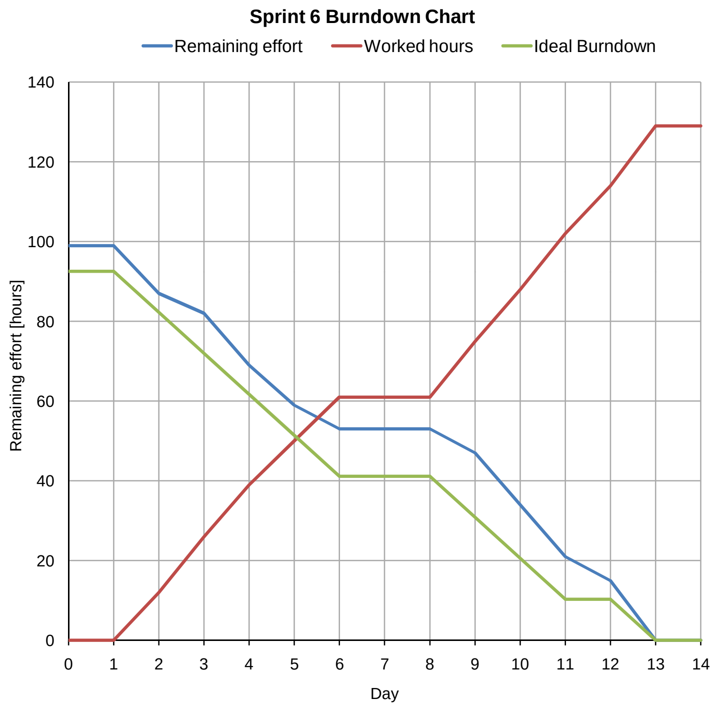
							Estimated hours remaining per day | Hours worked on task per day																												
Nr.	Priority	Status		Task	Verantwortlich	Hours worked	0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So
1	High		**Implementation**	Presence Engine	Ricardo Alvarez	41	40	40	7	33	7	26	7	19		19		19		19		19		19	6	12	6	6	8						
2	Medium			User Interface	Reto Brühwiler	66	50	50	8	42	7	35	7	28	8	20	6	14		14		14	6	8		30	8	22	8	16	8	8		8	
3	Medium			Action Plugin	Ricardo Alvarez	24	20	20		20		20		20	6	15	6	10		10		10	6	4	6										
						0																													
4	Medium		**Organisatorisches**	Meeting	Team	3	3	3		3	3																								
						0																													
						0																													
						0																													
						0																													
						0																													
						0																													
					Estimated work remaining:		**113**	**113**		**98**		**81**		**67**		**54**		**43**		**43**		**43**		**31**		**42**		**28**		**16**		**8**		**8**	
					Remaining working hours:		**103**	**103**		**93**		**82**		**72**		**62**		**51**		**51**		**51**		**41**		**31**		**21**		**10**		**0**		**0**	
					Hours worked per Day:				**15**		**17**		**14**		**14**		**12**						**12**		**12**		**14**		**16**		**8**				
Sprint finished					**Hours worked total:**	**134**		**0**	**15**	**15**	**32**	**32**	**46**	**46**	**60**	**60**	**72**	**72**	**72**	**72**	**72**	**72**	**84**	**84**	**96**	**96**	**110**	**110**	**126**	**126**	**134**	**134**	**134**	**134**	**134**

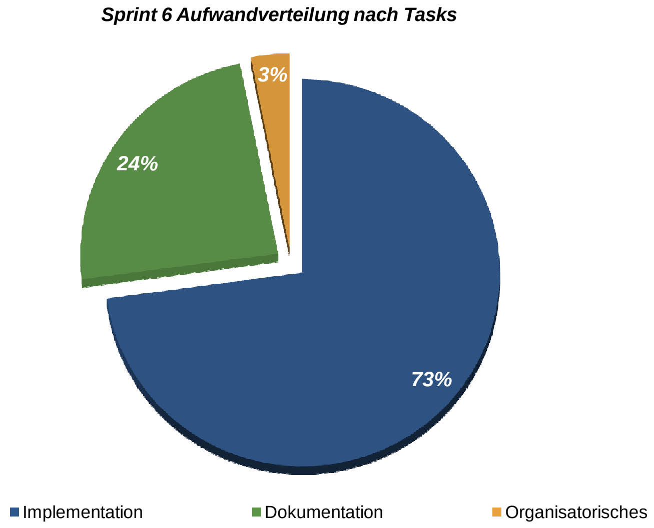




Sprint 6 Backlog

							Estimated hours remaining per day | Hours worked on task per day																												
Nr.	**Priority**	**Status**		**Task**	**Verantwortlich**	**Hours worked**	**0**	**1**	**Mo**	**2**	**Di**	**3**	**Mi**	**4**	**Do**	**5**	**Fr**	**6**	**Sa**	**7**	**So**	**8**	**Mo**	**9**	**Di**	**10**	**Mi**	**11**	**Do**	**12**	**Fr**	**13**	**Sa**	**14**	**So**
1	Medium		**Implementation**	User Interface	Reto Brühwiler	57	30	30	8	22	7	23	7	16	6	10	5	10		10		10	6	12	2	10	5	5	5	6	6				
2	Medium			Eventschnittstelle	Ricardo Alvarez	17	15	15	4	11	7	5	3	2	3																				
3	Medium			Persistierung	Team	20	20	20		20		20	3	17	2	15	5	10		10		10		10	5	5		5	5						
						0																													
5	Medium		**Dokumentation**	SAD	Team	31	30	30		30		30		30		30		30		30		30	8	22	6	16	7	10	2	8	8				
						0																													
7	Medium		**Organisatorisches**	Meeting	Team	4	4	4		4		4		4		4	1	3		3		3		3		3	2	1		1	1				
						0																													
						0																													
						0																													
						0																													
					Estimated work remaining:		**99**	**99**		**87**		**82**		**69**		**59**		**53**		**53**		**53**		**47**		**34**		**21**		**15**					
					Remaining working hours:		**93**	**93**		**82**		**72**		**62**		**51**		**41**		**41**		**41**		**31**		**21**		**10**		**10**		**0**		**0**	
					Hours worked per Day:				**12**		**14**		**13**		**11**		**11**						**14**		**13**		**14**		**12**		**15**				
Sprint finished					**Hours worked total:**	**129**		**0**	**12**	**12**	**26**	**26**	**39**	**39**	**50**	**50**	**61**	**61**	**61**	**61**	**61**	**61**	**75**	**75**	**88**	**88**	**102**	**102**	**114**	**114**	**129**	**129**	**129**	**129**	**129**



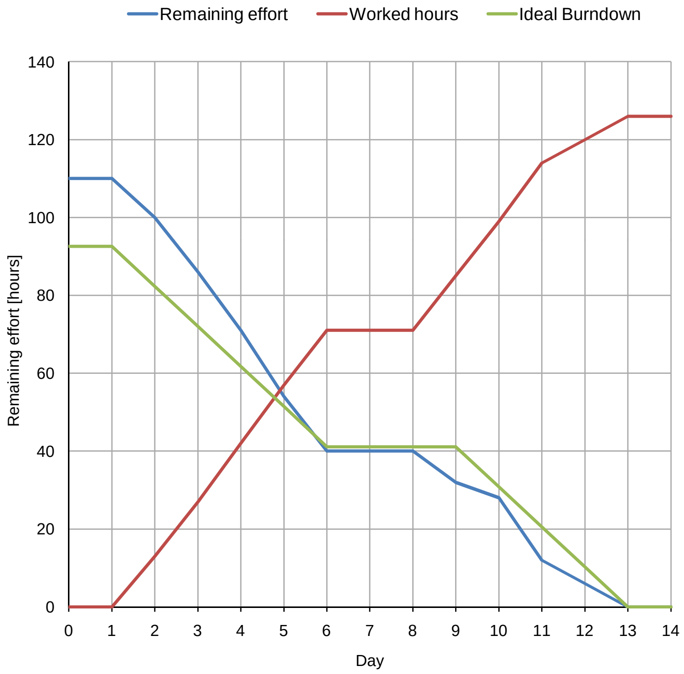


Sprint 7 Backlog

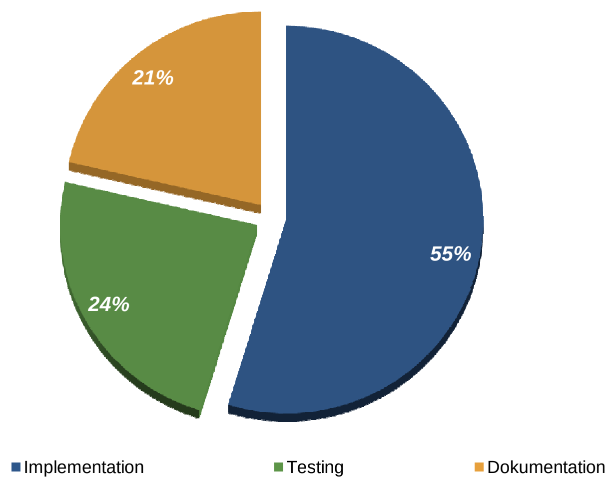
Nr.	Priority	Status		Task	Verantwortlich	Hours worked	0	1	Mo	2	Di	3	Mi	4	Do	5	Fr	6	Sa
							Estimated hours remaining per day | Hours worked on task per day												
1	Medium		**Implementation**	User Interface	Reto Brühwiler	26	20	20	6	14	6	8	2	6	4	2	2		
2	Medium			Persistierung	Team	13	10	10	5	8	4	4		4	4				
3	Medium			Code bereinigung	Team	30	20	20	2	18		18	4	14		14	2	12	
						0													
4	Medium		**Testing**	Testing	Team	30	30	30		30	4	26	4	22	3	18	6	12	
						0													
5	Medium		**Dokumentation**	External Designen	Reto Brühwiler	5	5	5		5		5	5						
6	Medium			Bereinigung	Team	18	20	20		20		20		20		20	4	16	
7	Medium			Testdokumentation	Ricardo Alvarez	4	5	5		5		5		5	4				
						0													
						0													
					Estimated work remaining:		**110**	**110**		**100**		**86**		**71**		**54**		**40**	
					Remaining working hours:		**93**	**93**		**82**		**72**		**62**		**51**		**41**	
					Hours worked per Day:				**13**		**14**		**15**		**15**		**14**		
Sprint finished					**Hours worked total:**	**126**		**0**	**13**	**13**	**27**	**27**	**42**	**42**	**57**	**57**	**71**	**71**	**71**

Nr.	Task	7	So	8	Mo	9	Di	10	Mi	11	Do	12	Fr	13	Sa	14	So
1	User Interface					6	3	3	3								
2	Persistierung																
3	Code bereinigung	12		12	4	8	3	15	3	12	6	6	6				
4	Testing	12		12	4	8	6	2	3								
5	External Designen																
6	Bereinigung	16		16	6	10	2	8	6								
7	Testdokumentation																
	Estimated work remaining:	**40**		**40**		**32**		**28**		**12**		**6**					
	Remaining working hours:	**41**		**41**		**41**		**31**		**21**		**10**		**0**		**0**	
	Hours worked per Day:				**14**		**14**		**15**		**6**		**6**				
Sprint finished	**Hours worked total:**	**71**	**71**	**71**	**85**	**85**	**99**	**99**	**114**	**114**	**120**	**120**	**126**	**126**	**126**	**126**	**126**

Sprint 7 Burndown Chart



Sprint 7 Aufwandverteilung nach Tasks



Anhang F

Risiko Analyse

Risikoanalyse

Risiko-Nr	Risikotitel	Risikobeschreibung	max. Schaden [h]	Eintritts-wahrscheinlichkeit	gewichteter Schaden [h]	Massnahmen zur Vermeidung/Verminderung	Vorgehen bei Eintreffen
R1	Ausfall Arbeitsstation	HW eines Projektmitgliedes fällt aus.	5	1.00 %	0.05	Regelmässig Einchecken / Backup vom Arbeitsrechner erstellen.	Daten von Backup oder SVN restoren, auf anderem Rechner weiterarbeiten.
R2	Ausfall HSR Infrastruktur	SVN-Server oder Netzwerk Infrastrkutur der HSR fällt aus.	5	1.00 %	0.05	Kopien auf Arbeitsrechner aktuell halten. Regelmässig auch die Arbeiten Anderer auf den eigenen Rechner spiegeln (SVN Update).	Teammitglieder synchronisieren sich auf anderem Weg.
R3	Datenverlust	Erarbeitete Projekt-Artefakte werden unwiderruflich geändert oder gelöscht	100	1.00 %	1.00	Arbeiten mit Versionsverwaltung, Backup Kopien auf allen Arbeitsstationen, zusätzliches Backup neben SVN.	Dateien aus letztem Backup wiederherstellen.
R4	Ausfall eines Projektmitgliedes	Ausfall eines Projektmitgliedes aufgrund Krankheit, Unfall, Studiumabbruch	100	10.00 %	10.00	Für zugeteilte Verantwortlichkeiten einen Stellvertreter vorsehen.	Arbeit innerhalb des bestehenden Teams aufteilen, Eventuell Funktionsumfang kürzen.
R5	Fehleinschätzung des Aufwandes	Zeitplan wird nicht eingehalten, da der Aufwand einzelner Aufgaben falsch eingeschätzt wurde.	50	10.00 %	5.00	Vorzeitig Problem erkennen und Massnahmen einleiten.	Mit dem Betreuer über eventuelle Kürzungen sprechen und gegebenenfalls Arbeitspackete anpassen.
R6	Spannungen im Team	Uneinigkeit im Team	100	5.00 %	5.00	Probleme und Bedenken ansprechen.	Gespräch im Team wenn das nicht hilft Betreuer als Vermittler einschalten
R7	Inkompartibilität	Getrennt entwickelte Komponenten arbeiten nicht zusammen	50	10.00 %	5.00	Schnittstellen genau definieren	Überprüfen der Schnittstellen; Interface/Converter dazwischen hängen; Schlimmstenfalls eine Komponente ersetzen/neu entwickeln
R8	Fehlerhafte Komponenten	Fehler in verwendeten nicht selbst entwickelten Komponenten	50	5.00 %	2.50	Neuste Version verwenden. Möglicht weit verbreitete Komponenten benutzen.	Recherche im Internet nach BugFix oder Workaround; Komponente ersetzen
R9	State Machine Effizienz	State Machine nicht ausreichend effizient	200	10.00 %	20.00	Genaue Evaluation der vorhandenen Technologien.	State Machine selber implementieren.
Summe			**660**	**53.00 %**	**48.6**		