HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

ins institute for networked solutions

Roboter Controller Showcase

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2010

Autoren:	Stefan Züger, Tobias Zürcher
Betreuer:	Prof. Hansjörg Huser
Projektpartner:	Zühlke Engineering AG, Schlieren
Experte:	Stefan Zettel, Ascentiv AG, Zürich
Gegenleser:	Prof. Beat Stettler

1	**Summary** • Abstract • Management Summary
2	**Technischer Bericht** • Einleitung & Übersicht • Anforderungen • Analyse • Architektur & Design • Ergebnisse & technische Erkenntnisse • Schlussfolgerungen
3	**Aufgabenstellung** • Aufgabenstellung HSR • Aufgabenstellung Zühlke Engineering Teil I • Aufgabenstellung Zühlke Engineering Teil II
4	**Requirements Analyse** • Interview • Software Requirements Specification
5	**Projektmanagement** • Projektplan • Iterationsdokumentation • Erfahrungsberichte • Sitzungsprotokolle
6	**Test** • Testdokumentation
7	**Quick Start Guides** • Benutzeranleitung Roboter- & Remote Controller • Installationsanleitung Entwicklungsumgebung • DPWS Entwicklungsdokumentation
8	**Präsentationen** • Zwischenpräsentation für Gegenleser • Präsentation Prototyp • Präsentation Beta
9	**Anhang** • Glossar • Literaturverzeichnis • Silverlight Error Codes • DPWS Spezifikation
10	**DVD** • DVD mit Projektdokumentation/Source Code • Installations-DVD für Entwicklungsumgebung

Summary

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

1 Abstract

Ausgangslage
Die Firma Zühlke Engineering AG möchte eine neue Schulungs-/Showcase-Plattform für Windows Mobile/Embedded aufbauen. Zur Demonstration der Plattform soll ein Roboter Controller auf einem Windows Embedded System entwickelt werden, welcher einen kleinen Roboter steuert und zusätzlich die Steuerfunktionalität für Windows Mobile Geräte über Web Services zur Verfügung stellt. Für die Web Services Kommunikation zwischen den Geräten soll eine C# Implementierung des DPWS Standards (Devices Profile for Web Services) realisiert werden. Das UI auf dem Embedded System soll in Silverlight for Embedded umgesetzt werden, damit neue GUI Konzepte getestet werden können.

Resultate
Im Laufe der Bachelorarbeit wurden die beiden Komponenten Roboter Controller und Remote Controller entwickelt. Der von Zühlke entwickelte Robotertreiber wurde im Roboter Controller integriert, so dass alle Funktionen des Roboters über das lokale Touchpanel oder die angeschlossene Tastatur gesteuert werden können. Des Weiteren wurde für die Kommunikation über Web Services zwischen Remote Controller und Roboter Controller der komplette DPWS-Stack des .NET Micro Frameworks aufs Compact Framework portiert. Das entwickelte DPWS Visual Studio Add-in ermöglicht eine komfortable Generierung der Proxys/Stubs. Die Auftraggeberin Zühlke erhält nach Abschluss der Arbeit eine über alle Layer und Tiers komplette Showcase-Plattform, welche sie für Weiterentwicklungen, Schulungen und Präsentationen bei Kunden einsetzen kann.

Technologien
Der Roboter Controller wurde für Windows Embedded CE 6.0 R3 in C# fürs Compact Framework 3.5 entwickelt. Das GUI ist wie gefordert mit Silverlight for Embedded in C++ realisiert. Beim Mobile Client handelt es sich um eine C# WinForms Compact Framework 3.5 Anwendung, welche auf einem Windows Mobile 6.5 Gerät zum Einsatz kommt.

2 Management Summary

2.1 Ausgangslage

Die Firma Zühlke Engineering AG möchte gerne eine auf den neusten Technologien basierte Windows Mobile/Embedded Schulungs- und Showcase Plattform. Die zu entwickelnde Plattform dient in Zukunft als:

- Schulungsplattform für interne Schulungen
- Showcase für Demonstrationen bei Kunden
- Plattform zum Testen neuer GUI- & Kommunikationskonzepte

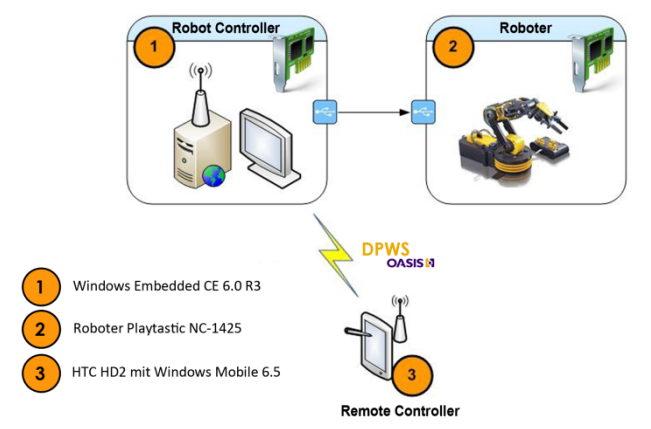


Abbildung 1: Systemübersicht der Showcase Plattform

2.2 Vorgehen/Technologien

Das Ausarbeiten einer konsistenten Anforderungsspezifikation war ein grosser Bestandteil der Aufgabenstellung. Als Software Entwicklungsprozess wurde iterativ inkrementell nach RUP vorgegangen. In einer risikominimierenden Projektplanung konnte bis zur Mitte des Projekts ein umfangreicher Architekturprototyp erstellt werden, mit dem alle technischen Risiken abgedeckt wurden. In den restlichen Wochen konnte in mehreren iterativen Konstruktionsphasen laufend die geforderten Features umgesetzt werden. Folgende Technologien wurden zur Umsetzung dieser Bachelorarbeit eingesetzt:

- C#
- C++
- Compact Framework 3.5 auf Windows Mobile 6.5
- Silverlight for Embedded auf Windows Embedded CE 6.0 R3

2.3 Ergebnisse

Zur Demonstration der Plattform wurden zwei Anwendungen entwickelt. Zum einen handelt es sich dabei um den Roboter Controller, der für das Windows Embedded CE 6.0 Betriebssystem entwickelt wurde, mit dem ein kleiner Roboter gesteuert werden kann. Mithilfe von Web Services stellt der Roboter Controller einige Funktionen zum Steuern des Roboters anderen Geräten zur Verfügung. Die zweite entwickelte Anwendung ist der Remote Controller auf einem Windows Mobile 6.5 Gerät, über welchen man zu beliebigen Roboter Controllern verbinden und den Roboter über Web Services steuern kann. Des Weiteren sind mehrere Roboter Controller in der Lage, sich selbstständig zu einem Robotern netz zusammen zu schliessen um gewisse Nachrichten untereinander auszutauschen.

Abbildung 2: Roboter Controller Applikation auf Touch Panel

Für die Kommunikation zwischen Remote Controller und Roboter Controller wurde die DPWS (Devices Profile for Web Services) Implementierung des .NET Micro Frameworks auf das Compact Framework 3.5 portiert. Der DPWS Standard ermöglicht mobilen Geräten über Web Services zu kommunizieren und beliebige Geräte dynamisch ohne zentralen Server in einem Netz aufzufinden.

Zusätzlich zum DPWS-Stack wurde der DPWS Codegenerator portiert und mit einem Visual Studio Add-in in die IDE integriert.

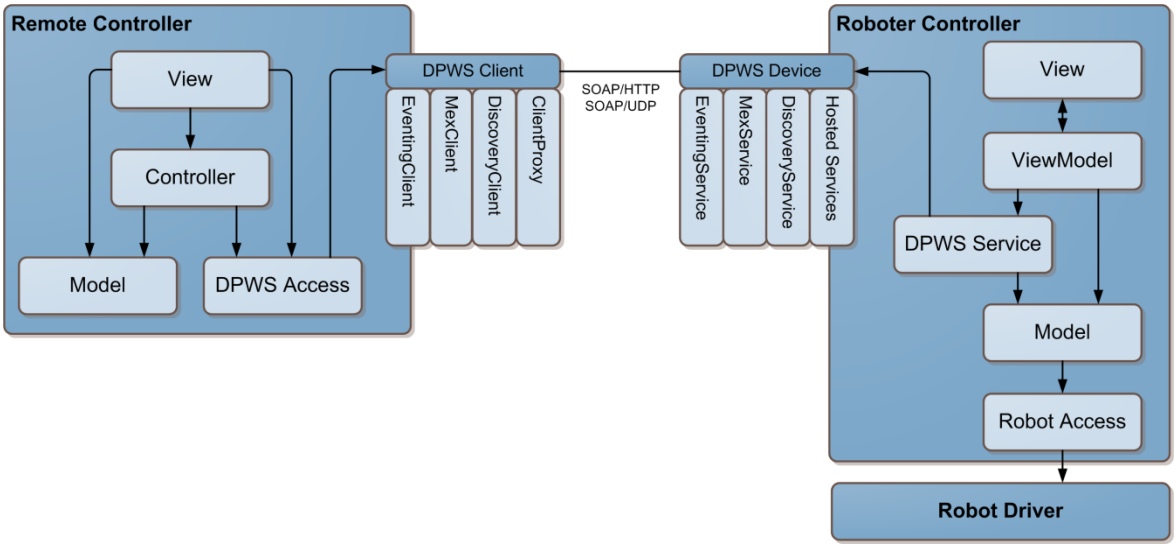


Abbildung 3: Übersicht Systemarchitektur von Remote Controller und Roboter Controller

2.4 Ausblick

Zühlke verwendet die entwickelte Showcase Plattform nach Abschluss der Bachelorabeit wie geplant als Demonstrationsobjekt bei Kunden und zur internen Schulung ihrer Mitarbeiter. Der portierte DPWS-Stack wird direkt zur Umsetzung von Kundenprojekten weiterverwendet.

Technischer Bericht

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

Review

Datum	Kommentar	Revisor
24.04.2009	Review der DPWS Analyse	TZ
12.05.2010	Review der Dokumentation zu Roboter Controller und Remote Controller	SZ
10.06.2010	Review Architektur und Design Kapitel	TZ
15.06.2010	Review der Schlussfolgerungen	SZ

Tabelle 1: Reviews

Teil 1: Einleitung und Übersicht

1 Projektauftrag

1.1 Einleitung

Für den Aufbau einer Schulungsplattform soll eine Robotersteuerung entwickelt werden, mit der ein Spielzeugroboter gesteuert werden kann. Die Schulungsplattform verwendet Zühlke intern für die Aus- und Weiterbildung zu technologiespezifischen Themen betreffend Windows CE und Windows Mobile. Weiterhin dient die Plattform zum Testen neuer GUI Technologien und Kommunikationskonzepten.

Teile der Gesamtapplikation sollen auch als Showcase für die Demonstrationen bei Kunden verwendet werden.

1.2 Aufgabenstellung

Im Wesentlichen besteht der Showcase aus zwei Softwarekomponenten: dem Roboter Controller (1) und dem Remote Controller (3). Der Roboter wird über eine Treiberschnittstelle (2) gesteuert, welche von Zühlke geliefert wird.

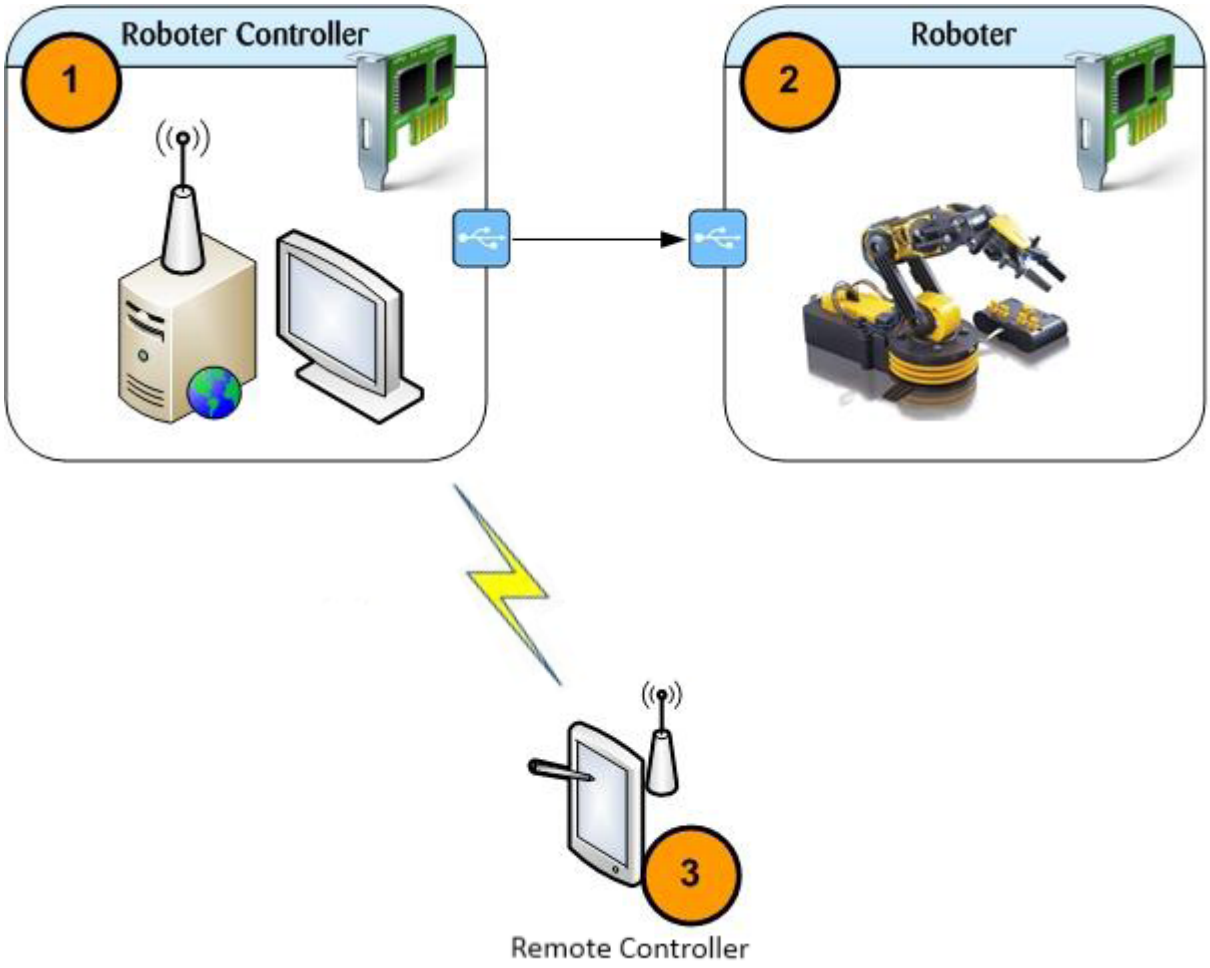


Abbildung 1: Projektscope

Der Roboter Controller ist die zentrale Steuereinheit für den Roboter, welcher auf einer Windows Embedded CE 6.0 R3 Plattform betrieben wird. Der Roboter kann direkt am Roboter Controller über einen Touchscreen gesteuert werden. Des Weiteren bietet der Controller Services zur Steuerung des Roboters über das Netzwerk an. Der Remote Controller besteht aus einem GUI für die Fernsteuerung und einem Webservice Client für die

Kommunikation mit dem Roboter Controller. Die Kommunikation zwischen Remote Controller und Roboter Controller basiert auf dem DPWS Standard, welcher in managed Code implementiert werden muss. Die Softwareentwicklung bezieht sich ausschliesslich auf den DPWS-Stack und die Software Applikationen für den Roboter Controller und den Remote Controller.

1.2.1 Kommunikationskonzept: Devices Profile for Web Services

Die Kommunikation zwischen Roboter- und Remote Controller soll mit Devices Profile for Web Services (kurz DPWS) erfolgen. Für die embedded Entwicklung gibt es derzeit lediglich eine native Implementation des DPWS-Stack. Das .NET Micro Framework verfügt über eine managed Implementierung, welche unter der Apache 2.0 Lizenz *[URL1.1]* veröffentlicht wurde. Innerhalb der Arbeit soll unter anderem ein DPWS-Stack entwickelt werden, den Zühlke in Zukunft für die Implementierung von Web Services in anderen Projekten verwenden kann.

Die Implementation des DPWS-Stack wird mit den beiden Beispielapplikationen (Roboter Controller und Remote Controller) getestet. Weitere Details zur DPWS Implementation können im Kapitel *8 Devices Profile for Web Services (DPWS)* nachgeschlagen werden. Eine Analyse zum DPWS ist im Kapitel *6 Analyse DPWS* dokumentiert.

1.2.2 Testen neuer GUI Konzepte: Silverlight for Embedded

Microsoft veröffentlichte im Oktober 2009 (3 Monate vor Beginn Bachelorarbeit) Windows Embedded CE 6.0 R3, welches erstmals eine Implementation von Silverlight für Embedded Systeme beinhaltet. Silverlight for Embedded besitzt grundsätzlich die gleichen Features wie das gewöhnliche Silverlight 2, jedoch mit der Besonderheit, dass der Code-Behind in C++ geschrieben wird. Trotzdem soll der C++ Layer dünn gehalten werden, so dass möglichst viel mit C# ausprogrammiert werden kann. Eine spezielle Herausforderung hierbei ist die Kommunikation zwischen managed- und nativem Code, da zum Beispiel kein Databinding vorhanden ist. Betrachtet man das MVVM Pattern, welches bei Silverlight/WPF Anwendungen sehr populär ist, ist der View-Layer eine C++ DLL, welche von einem managed Assembly geladen wird. Weitere Details dazu sind im Kapitel *9.3 Silverlight for Embedded* dokumentiert.

1.2.3 DPWS Visual Studio Integration

Das Generieren von Proxy, Stubs und WSDL Definition soll in Visual Studio integriert sein. Dabei gibt es zwei Ansätze: Der „Contract First“ Ansatz, bei dem zuerst das WSDL geschrieben wird und daraus die Proxys und Stubs generiert werden. Viele Entwickler bevorzugen das Attributieren von Interface und Klassen. Anhand der Attributierungen kann die WSDL Definition generiert werden, wodurch wiederum die Proxys und Stubs generiert werden können.

Hierzu wird ein Add-in für Visual Studio implementiert, welches die Generierung über ein Kontextmenü an diversen Orten wie zum Beispiel im Solution Explorer oder im Code Window einer WSDL-Datei ermöglicht.

Teil 2: Anforderungen

2 Einleitung Anforderungen

Das Erarbeiten der Anforderungsspezifikation war ein wichtiger Bestandteil dieser Bachelorarbeit. Nach dem ersten Kickoff-Meeting mit dem Auftraggeber, hatte das Erstellen einer konsistenten Software Requirements Specification oberste Priorität. Aus diesem Grund wurde in der Inception Phase gleich ein Interviewtermin vereinbart um in einem Gespräch die genauen Anforderungen zu erfragen. Darauf konnte eine Software Requirements Specification erstellt werden, welche dem Auftraggeber mehrmals zum Review vorgelegt wurde, bevor eine endgültige Version entstand.

Dieser Teil des technischen Berichts fasst die erstellte Anforderungsspezifikation kurz zusammen, damit Klarheit für die weiteren Teile geschaffen wird. Die komplette Anforderungsspezifikation kann im Kapitel *4 Requirements Analyse* nachgeschlagen werden.

3 Anforderungen

Nach dem Interviewgespräch mit dem Auftraggeber war direkt klar, dass die Anforderungen in drei Komponenten aufgeteilt werden können. Die Komponenten sind DPWS, Roboter Controller und Remote Controller. Nachfolgend werden die Anforderungen an die Komponenten jeweils mit einem Use Case Diagramm zusammengefasst.

3.1 Anforderungen DPWS

Unter der Komponente DPWS wird verlangt, dass der DPWS 1.1 Standard von OASIS in C# umgesetzt wird. Dabei müssen alle 5 Teile von DPWS (Discovery, Description, Messaging, Eventing & Security) implementiert werden, wobei die Security Komponente die niedrigste Priorität hat.

Wie das folgende Use Case Diagramm zeigt, muss zusätzlich für den Entwickler ein Code Generator (analog WCF SvcUtil) erstellt werden, mit dem er anhand einer WSDL Datei die passenden Proxys/Stubs oder anhand eines Assemblys die WSDL Datei generieren kann.

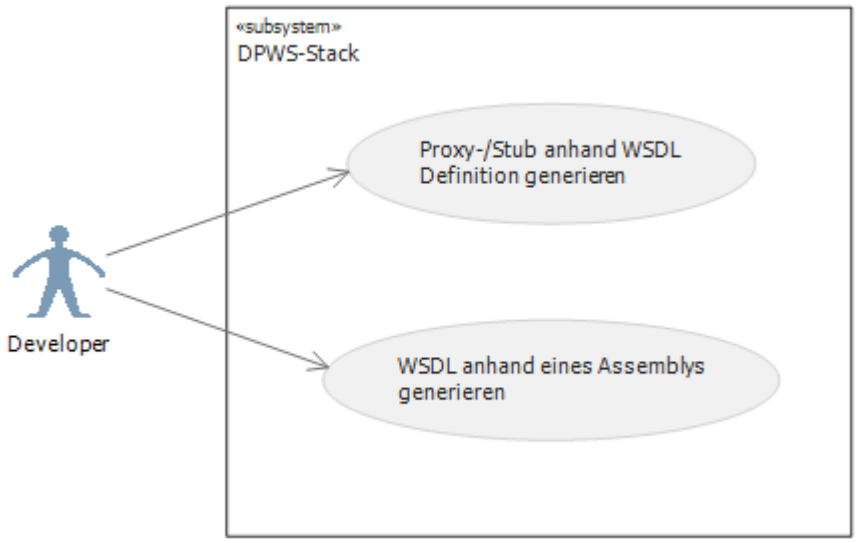


Abbildung 2: Use-Case Diagramm "Subsystem DPWS-Stack"

Des Weiteren soll ein Visual Studio Add-in geschrieben werden, mit dem der Code Generator direkt aus dem Visual Studio aufgerufen werden kann.

3.2 Anforderungen Roboter Controller

Folgendes Use Case Diagramm zeigt alle funktionalen Anforderungen an die Komponente Roboter Controller, welche auf einem Windows Embedded CE 6.0 R3 System betrieben wird.

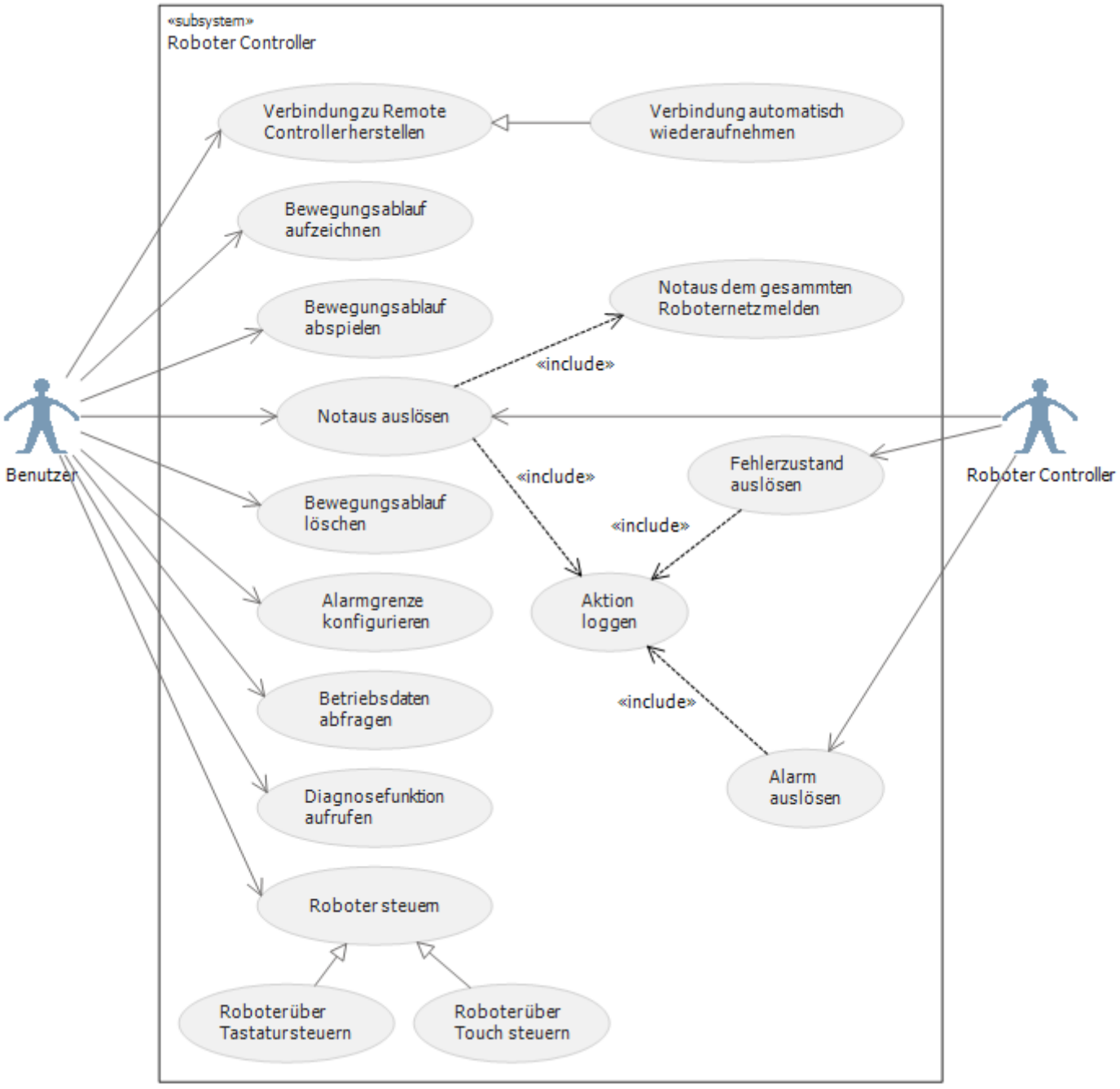


Abbildung 3: Use-Case Diagramm "Subsystem Roboter Controller"

Die Komponente Roboter Controller beinhaltet jegliche Funktionalität, welche auf dem Windows Embedded System ausgeführt werden kann. Speziell ist hierbei zu erwähnen, dass das GUI der Roboter Controller Applikation mit Silverlight for Embedded in C++ umgesetzt werden muss.

3.3 Anforderungen Remote Controller

Beim Remote Controller handelt es sich um eine Windows Mobile 6.5 Anwendung, welche genutzt wird, um den Roboter über eine WLAN Verbindung zum Roboter Controller zu steuern.

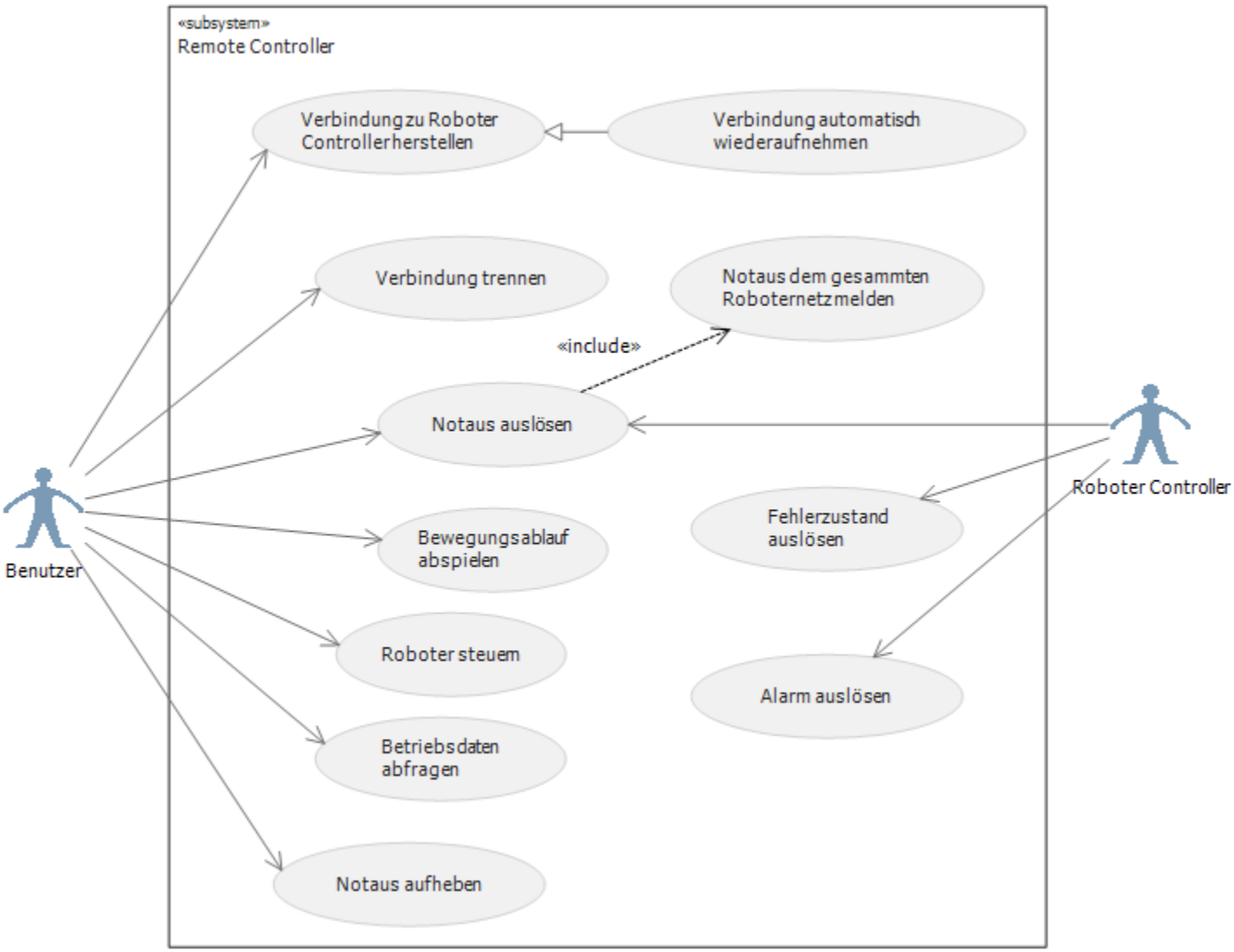


Abbildung 4: Use-Case Diagramm "Subsystem Remote Controller"

Teil 3: Analyse

4 Einleitung Analyse

Als Erstes wird die Domainanalyse beschrieben. Mit einem Domain Modell wird eine erste Übersicht über die Problem Domain verschafft. Zusätzlich werden die Zustände des Roboters anhand eines State Diagramms visualisiert.

Der zweite Teil befasst sich mit der Analyse des DPWS Standards.

4.1 Referenzen

Folgende Dokumente wurden als Basis für die Analyse verwendet:

Referenz	Quelle
Software Requirements Specification	/04_Requirements Analyse /Software_Requirements_Specification.pdf
DPWS Spezifikation	/90_Technische Literatur/wsdd-dpws-1.1-spec.pdf

Tabelle 2: Referenzen

5 Domainanalyse

5.1 Domain Model

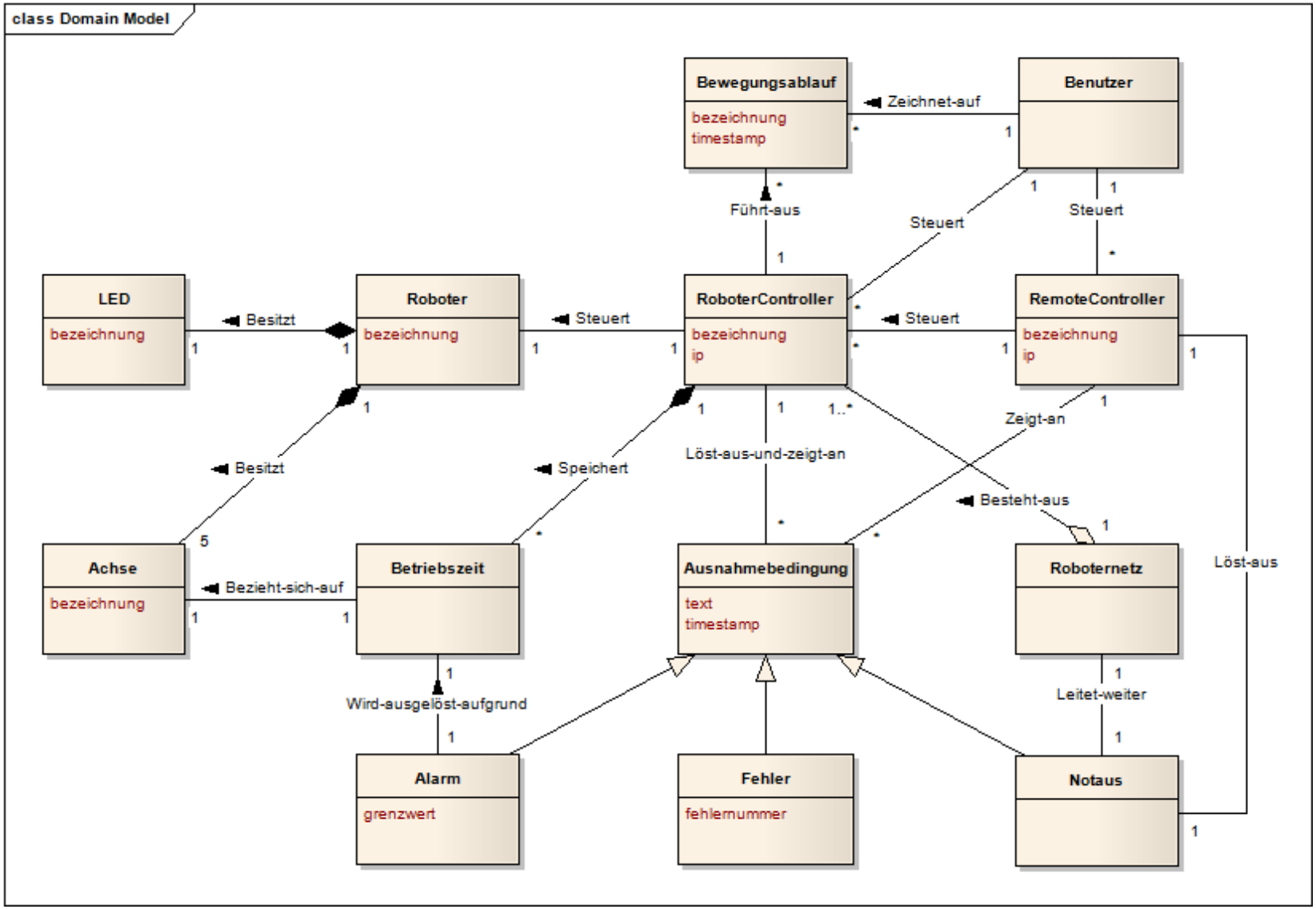


Abbildung 5: Domain Model

5.1.1 Konzeptbeschreibung

Roboter		
Beschreibung	Der Roboter welcher gesteuert wird.	
Attribute	bezeichnung	Bezeichnung zur Identifizierung des Roboters
Beziehung	Der Roboter besitzt ein LED Der Roboter besitzt 5 Achsen.	

LED		
Beschreibung	Das LED des Roboters.	
Attribute	bezeichnung	Bezeichnung zur Ansteuerung der LED
Beziehung	Das LED gehört zum Roboter.	

Achse		
Beschreibung	Eine Achse des Roboters.	
Attribute	bezeichnung	Bezeichnung zur Ansteuerung der Achse
Beziehung	Die Achse gehört zum Roboter. Die Achse besitzt eine Betriebszeit.	

Betriebszeit		
Beschreibung	Die Betriebszeit speichert die Dauer, wie lange eine Achse in Betrieb ist.	
Attribute	-	-
Beziehung	Die Betriebszeit bezieht sich auf eine Achse. Die Betriebszeit ist mit einem Alarm assoziiert.	

Alarm		
Beschreibung	Sobald eine Betriebszeit einen gewissen Wert überschreitet wird ein Alarm ausgelöst.	
Attribute	-	-
Beziehung	Der Alarm wird aufgrund der Betriebszeit ausgelöst. Der Alarm ist eine Spezialisierung einer Ausnahmebedingung.	

RoboterController		
Beschreibung	Der RoboterController nimmt Befehle entgegen und führt diese am Roboter aus. Des Weiteren führt dieser die Betriebszeiten pro Achse und kann komplette Bewegungsabläufe abspielen.	
Attribute	bezeichnung	Bezeichnung zur Ansteuerung des Roboter-Controller
	ip	IP Adresse
Beziehung	Der RoboterController führt einen Bewegungsablauf aus. Der RoboterController steuert den Roboter. Der RoboterController speichert die Betriebszeiten der Achsen. Der RoboterController löst Ausnahmebedingungen aus und zeigt sie an. Der RoboterController nimmt Befehle vom RemoteController entgegen. Der RoboterController gehört zu einem Roboternetz. Der RoboterController wird von einem Benutzer gesteuert.	

RemoteController		
Beschreibung	Der RemoteController dient zur Steuerung des Roboters per Remote, welcher jedoch nicht direkt mit dem Roboter kommuniziert, sondern via RoboterController die Befehle an den Roboter sendet.	
Attribute	bezeichnung	Identifizierung des RemoteController
	ip	IP Adresse
Beziehung	Der RemoteController sendet Befehle an den RoboterController Der RemoteController nimmt Befehle von einem Benutzer entgegen. Der RemoteController löst ein Notaus aus. Der RemoteController zeigt Ausnahmebedingungen an.	

Bewegungsablauf		
Beschreibung	Ein Bewegungsablauf repräsentiert mehrere aufeinanderfolgende Bewegungen, welche beliebig oft wieder abgespielt werde können.	
Attribute	bezeichnung	Bezeichnung zur Identifikation des Bewegungsablaufs
	timestamp	Zeitpunkt wann der Bewegungsablauf gespeichert wurde.
Beziehung	Der Bewegungsablauf wird vom RoboterController ausgeführt. Der Bewegungsablauf wird vom Benutzer aufgezeichnet.	

Benutzer		
Beschreibung	Der Benutzer steuert den Roboter über den RemoteController oder RoboterController.	
Attribute	-	-
Beziehung	Der Benutzer zeichnet Bewegungsabläufe auf. Der Benutzer steuert den RoboterController. Der Benutzer steuert den RemoteController.	

Roboternetz		
Beschreibung	Alle Roboter sind über einen LAN Anschluss mit einem Roboternetz verbunden, über welches die Roboter miteinander kommunizieren können (z.B. Notaus weitermelden).	
Attribute	-	-
Beziehung	Das Roboternetz besteht aus mehreren RoboterController. Das Roboternetz erhält Notausmeldungen vom RemoteController.	

Ausnahmebedingung		
Beschreibung	Ausnahmebedingungen sind Zustände oder Ereignisse, welche bestimmte Aktionen auslösen.	
Attribute	text	Details zur aufgetretenen Ausnahmebedingung
	timestamp	Zeitpunkt, an dem die Ausnahmebedingung ausgelöst wurde
Beziehung	Eine Ausnahmebedingung wird vom RoboterController oder RemoteController ausgelöst und angezeigt.	

Notaus		
Beschreibung	Ein Notaus ist eine Spezialisierung einer Ausnahmebedingung. Das Notaus wird dem ganzen Roboternetz mitgeteilt.	
Attribute	-	-
Beziehung	Ein Notaus wird ins Roboternetz weitergeleitet. Ein Notaus wird vom RemoteController ausgelöst.	

Fehler		
Beschreibung	Der Treiber des Roboters kann einen Fehler auslösen, welcher anschliessend dem Benutzer angezeigt wird, welcher diesen zuerst bestätigen muss.	
Attribute	fehlernummer	Nummer zur Identifikation des Fehlers.
Beziehung	Der Fehler ist eine Spezialisierung einer Ausnahmebedingung.	

5.2 State Diagramm Roboter

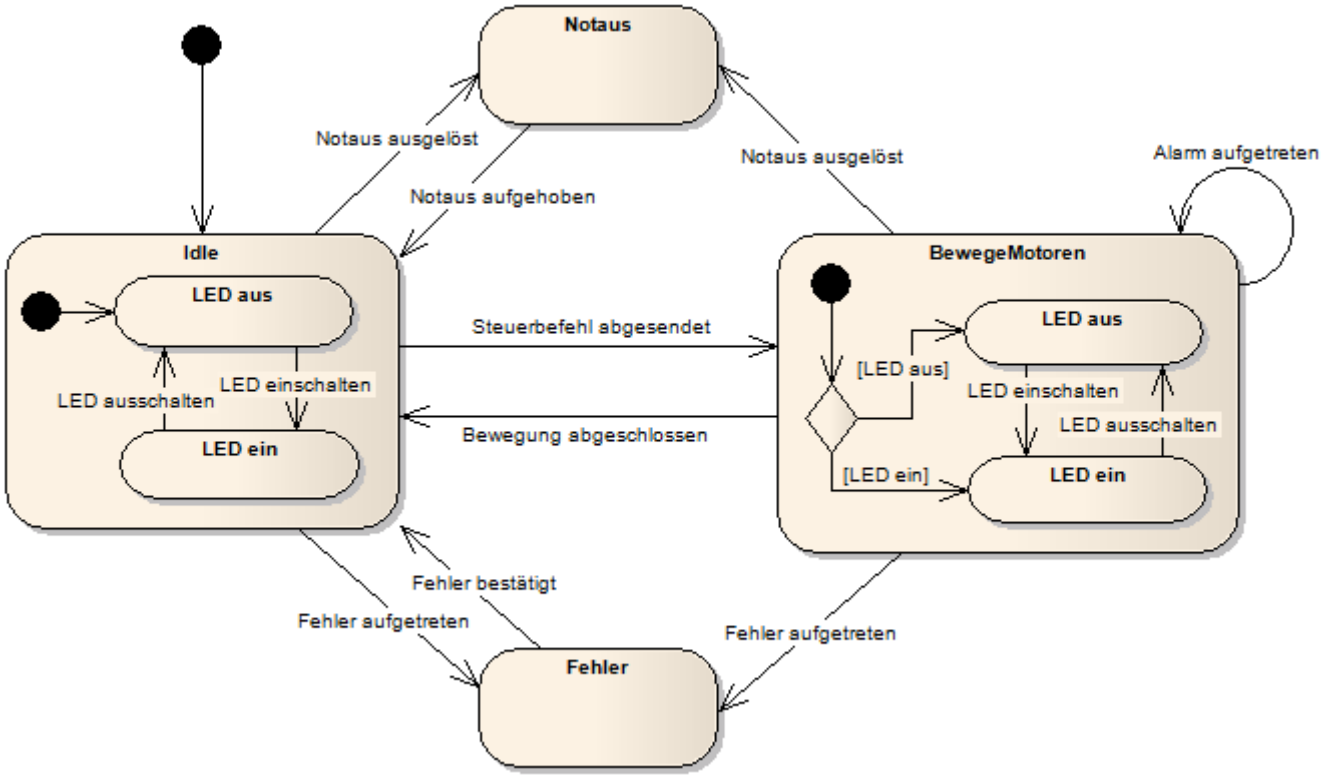


Abbildung 6: State Diagramm des Roboters

Das State Diagramm ist aus Sicht des Roboters modelliert.

6 Analyse DPWS

Gemäss Software Requirements Specifiation muss in dieser Bachelorarbeit eine Implementierung des DPWS Standards umgesetzt werden. Das folgende Kapitel beschreibt die Analyse der einzelnen Komponenten des Standards. Zum Schluss des Kapitels wird analysiert, welche Implementierungen des Standards bereits vorhanden sind und inwiefern in dieser Bachelorarbeit der Standard in C# umgesetzt werden soll.

Das Devices Profile for Web Services (DPWS) spezifiziert seit 2009 einen offiziellen OASIS Standard in der Version 1.1 *[OASIS]*, mit dem es ermöglicht werden soll, Web Services auf eingebetteten Systemen einzusetzen. Der DPWS Standard ist eine Kombination von verschiedenen existierenden WS-* Spezifikationen und wird mit ein paar wenigen Erweiterungen zu einem Profil zusammengefasst.

Das folgende Diagramm zeigt die Architektur des DPWS-Stacks und die darin verwendeten Standards:

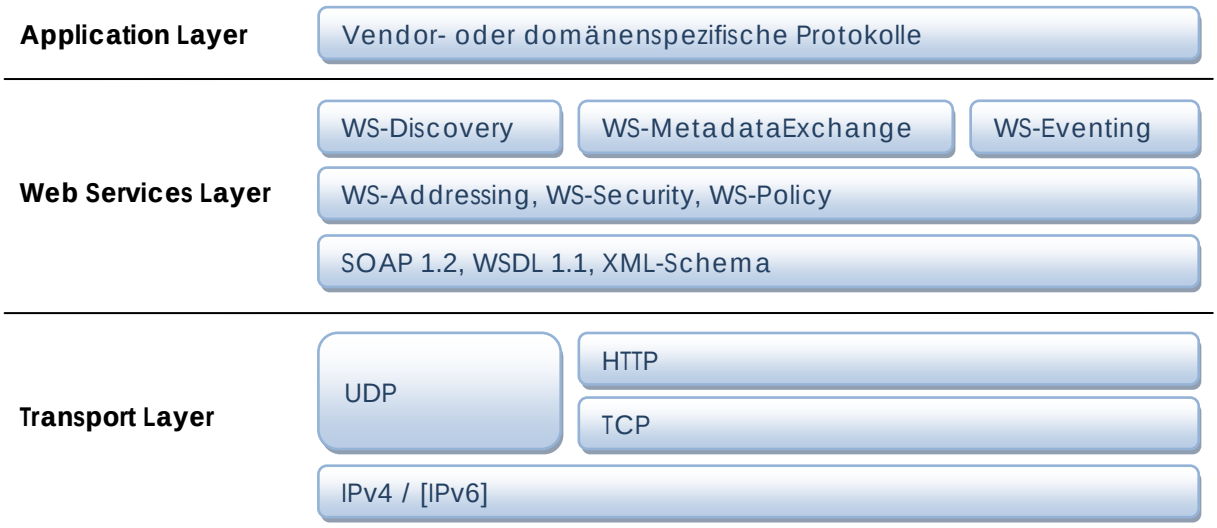


Abbildung 7: Architektur DPWS-Stack

6.1 Messaging

Die Messaging Komponente des DPWS Standards beschreibt die allgemeinen Voraussetzungen um Nachrichten übers Netz auszutauschen. Die ganzen Nachrichten werden in SOAP 1.2 Nachrichten verpackt, wobei die diese entweder per SOAP-over-HTTP oder SOAP-over-UDP versendet werden.

SOAP-over-UDP wird für alle Discovery Nachrichten (multicast) und SOAP-over-http (unicast) für die eigentlichen Service Calls verwendet. Für das Versenden von binären Daten über Web Services wird der MTOM Standard eingesetzt.

6.2 Discovery

Die Discovery Komponente des DPWS-Stacks beinhaltet ausschliesslich Teile des WS-Discovery Standards. Diese Komponente wird gebraucht, damit Geräte im Netz automatisch gefunden werden können. Im Discovery Teil des DPWS Standards werden folgende Nachrichten versendet:

- Hello
- Bye
- Probe
- ProbeMatches
- Resolve
- ResolveMatches

Das folgende Diagramm zeigt eine Übersicht über den zu erwartenden Discovery Netzwerkverkehr zwischen DPWS Host und Client.

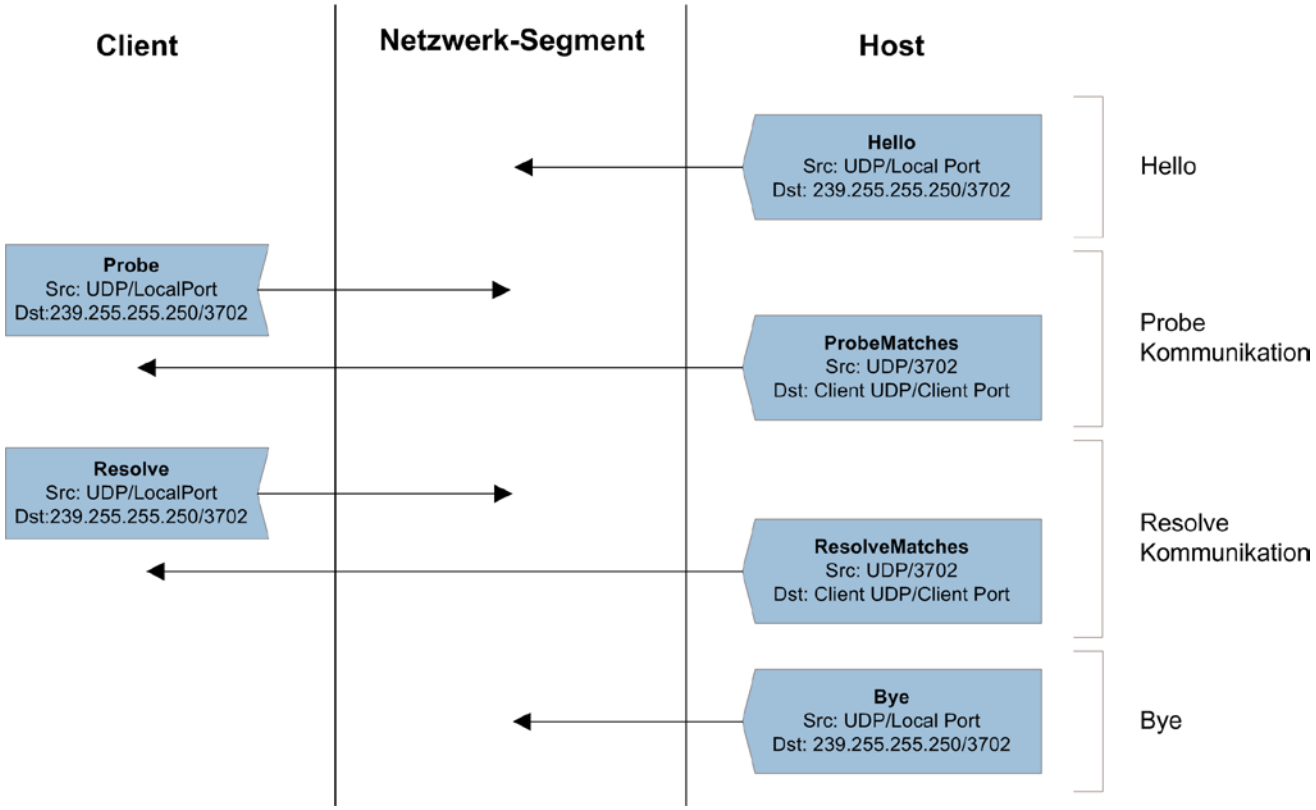


Abbildung 8: Netzwerkverkehr DPWS Discovery

6.2.1 Hello Nachricht

Die DPWS Hello Nachricht ist eine WS-Discovery Nachricht, welche benutzt wird um die Präsenz eines Gerätes im Netz anzukündigen. Jedes Gerät versendet eine solche Nachricht, wenn es gestartet wurde und bereit ist, seine Services anzubieten. Hello Nachrichten werden über UDP Multicasts ins ganze Subnetz gesendet.

```
<soap:Envelope
        xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
        xmlns:wsa='http://www.w3.org/2005/08/addressing'
        xmlns:wsdp='http://schemas.xmlsoap.org/ws/2006/02/devprof'
        xmlns:wsd='http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01'
        xmlns:rco='http://schemas.hsr.ch/RoboterController'>
  <soap:Header>
    <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
    <wsa:Action>
       http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Hello
    </wsa:Action>
    <wsa:MessageID>urn:uuid:634074bd-2884-4861-9417-4832f9ebc58d</wsa:MessageID>
    <wsd:AppSequence InstanceId='2'
       SequenceId='urn:uuid:c883e4a8-9af4-4bf4-aaaf-06394151d6c0' MessageNumber='7'/>
  </soap:Header>
  <soap:Body>
    <wsd:Hello>
      <wsa:EndpointReference>
        <wsa:Address>urn:uuid:2BC75C97-E266-4420-A66C-C7BD5EE5222A</wsa:Address>
      </wsa:EndpointReference>
      <wsd:Types>wsdp:Device rco:DeviceTypeRoboterController</wsd:Types>
      <wsd:XAddrs>
         http://152.96.233.53:8084/2BC75C97-E266-4420-A66C-C7BD5EE5222A
      </wsd:XAddrs>
      <wsd:MetadataVersion>1</wsd:MetadataVersion>
    </wsd:Hello>
  </soap:Body>
```

6.2.2 Probe Nachricht

Eine Probe Nachricht dient dem Finden von Services im Netzwerk anhand eines Servicetyps. Es gibt auch die Möglichkeit generell nach allen Services im Subnetz zu suchen, indem gar kein Servicetyp mitgegeben wird. Probe Nachrichten werden über UDP Multicasts ins ganze Subnetz gesendet.

```
<soap:Envelope
    xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
    xmlns:wsa='http://www.w3.org/2005/08/addressing'
    xmlns:wsd='http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01'
    xmlns:rco='http://schemas.hsr.ch/RoboterController'>
  <soap:Header>
    <wsa:To soap:mustUnderstand='1'>
     urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01
    </wsa:To>
    <wsa:Action soap:mustUnderstand='1'>
     http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Probe
    </wsa:Action>
    <wsa:MessageID>urn:uuid:9c607433-1784-41d3-bb4b-fbe4a8aa9d2a</wsa:MessageID>
  </soap:Header>
  <soap:Body>
    <wsd:Probe>
      <wsd:Types>rco:DeviceTypeRoboterController</wsd:Types>
    </wsd:Probe>
  </soap:Body>
</soap:Envelope>
```

6.2.3 ProbeMatches Nachricht

Die ProbeMatches Nachricht wird von einem DPWS Host als Antwort auf eine Probe Nachricht versendet. Die ProbeMatches Nachricht enthält alle Endpoint-Adressen des Hosts, welche mit dem angeforderten Servicetyp übereinstimmen. Zusätzlich kann der Host direkt die Adresse (XAddr) zum Endpoint mitliefern, damit der Client keine Resolve Nachricht mehr senden muss. ProbeMatches Nachrichten werden als UDP Unicast Datagramme versendet.

```
<soap:Envelope
  xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
  xmlns:wsa='http://www.w3.org/2005/08/addressing'
  xmlns:wsdp='http://schemas.xmlsoap.org/ws/2006/02/devprof'
  xmlns:wsd='http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01'
  xmlns:rco='http://schemas.hsr.ch/RoboterController'>
  <soap:Header>
    <wsa:To soap:mustUnderstand="1">
      http://www.w3.org/2005/08/addressing/anonymous
    </wsa:To>
    <wsa:Action soap:mustUnderstand="1">
      http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ProbeMatches
    </wsa:Action>
    <wsa:MessageID>urn:uuid:5fa30321-279d-4e1a-916f-37e5433aba93</wsa:MessageID>
    <wsa:RelatesTo>urn:uuid:9c607433-1784-41d3-bb4b-fbe4a8aa9d2a</wsa:RelatesTo>
    <wsd:AppSequence InstanceId='1270774107'
                     SequenceId='urn:uuid:c883e4a8-9af4-4bf4-aaaf-06394151d6c0'
                     MessageNumber='3'/>
  </soap:Header>
  <soap:Body>
    <wsd:ProbeMatches>
      <wsd:ProbeMatch>
        <wsa:EndpointReference>
          <wsa:Address>urn:uuid:2BC75C97-E266-4420-A66C-C7BD5EE5222A</wsa:Address>
        </wsa:EndpointReference>
        <wsd:Types>wsdp:Device rco:DeviceTypeRoboterController</wsd:Types>
        <wsd:XAddrs>
          http://10.0.0.107:8084/2BC75C97-E266-4420-A66C-C7BD5EE5222A
        </wsd:XAddrs>
        <wsd:MetadataVersion>1</wsd:MetadataVersion>
      </wsd:ProbeMatch>
    </wsd:ProbeMatches>
  </soap:Body>
```

6.2.4 Resolve Nachricht

Resolve Nachrichten werden von Clients versendet, um zu einer bereits bekannten Endpoint Adresse die dazu passende Transportadresse (XAddr) zu erhalten. Sofern der DPWS Host in der ProbeMatches Nachricht bereits eine XAddr mitgeliefert hat, benötigt es kein Resolve mehr. Resolve Nachrichten werden über UDP Multicasts ins ganze Subnetz gesendet.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery">
  <soap:Header>
    <wsa:To>
      urn:schemas-xmlsoap-org:ws:2005:04:discovery
    </wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2005/04/discovery/Resolve
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:38d1c3d9-8d73-4424-8861-6b7ee2af24d3
    </wsa:MessageID>
  </soap:Header>
  <soap:Body>
    <wsd:Resolve>
      <wsa:EndpointReference>
        <wsa:Address>
          urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
        </wsa:Address>
      </wsa:EndpointReference>
    </wsd:Resolve>
  </soap:Body>
</soap:Envelope>
```

6.2.5 ResolveMatches Nachricht

Die ResolveMatches Nachricht wird von einem DPWS Host als Antwort auf eine Resolve Nachricht versendet. Die ResolveMatches Nachricht liefert die zum angefragten Endpoint passende Transportadresse. ResolveMatches Nachrichten werden als UDP Unicast Datagramme versendet.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof">
  <soap:Header>
    <wsa:To>
      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2005/04/discovery/ResolveMatches
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:64ddd01c-b0d6-4afd-aba6-6f1f161ce9d4
    </wsa:MessageID>
    <wsa:RelatesTo>
      urn:uuid:38d1c3d9-8d73-4424-8861-6b7ee2af24d3
    </wsa:RelatesTo>
    <wsd:AppSequence InstanceId="1"
        SequenceId="urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772"
        MessageNumber="6">
    </wsd:AppSequence>
  </soap:Header>
  <soap:Body>
    <wsd:ResolveMatches>
      <wsd:ResolveMatch>
        <wsa:EndpointReference>
          <wsa:Address>
            urn:uuid:37f86d35-e6ac-4241-964f-1d9ae46fb366
          </wsa:Address>
        </wsa:EndpointReference>
        <wsd:Types>wsdp:Device</wsd:Types>
        <wsd:XAddrs>
          http://192.168.0.2:5357/37f86d35-e6ac-4241-964f-1d9ae46fb366
        </wsd:XAddrs>
        <wsd:MetadataVersion>2</wsd:MetadataVersion>
      </wsd:ResolveMatch>
    </wsd:ResolveMatches>
  </soap:Body>
</soap:Envelope>
```

6.2.6 Bye Nachricht

Die DPWS Bye Nachricht ist eine WS-Discovery Nachricht, welche benutzt wird, um dem Netzwerk mitzuteilen, wenn ein DPWS Gerät das Netz verlassen hat. Jedes DPWS Gerät versendet vor dem Verlassen des Netzes eine Bye Nachricht.

```
<soap:Envelope
    xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
    xmlns:wsd='http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01'>
  <soap:Header>
    <wsa:To> urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye</wsa:Action>
    <wsa:MessageID>urn:uuid:193ccfa0-347d-41a1-9285-f500b6b96a15</wsa:MessageID>
    <wsd:AppSequence InstanceId='2'
        SequenceId='urn:uuid:369a7d7b-5f87-48a4-aa9a-189edf2a8772'
        MessageNumber='21'>
    </wsd:AppSequence>
  </soap:Header>
  <soap:Body>
    <wsd:Bye>
      <wsa:EndpointReference>
        <wsa:Address>urn:uuid:2BC75C97-E266-4420-A66C-C7BD5EE5222A</wsa:Address>
      </wsa:EndpointReference>
    </wsd:Bye>
  </soap:Body>
</soap:Envelope>
```

6.3 Description

Die Description Komponente des DPWS Standards beschreibt, wie Geräte Informationen über sich zur Verfügung stellen und wie Clients diese Informationen abfragen können.

6.3.1 Metadata Exchange

Um an die Informationen eines Gerätes zu kommen, muss gemäss WS-MetadataExchange ein Metadaten Austausch stattfinden. Ein DPWS Gerät liefert über den MetadataExchange folgende Informationen:

- **ThisModel**: Gibt Informationen zum Gerätetyp. Informationen sind zum Beispiel Manufacturer, Modelname, Modelnummer
- **ThisDevice**: Gibt Informationen über das spezifische Gerät. Informationen sind zum Beispiel Friendly Name, Firmware Version, Seriennummer
- **Relationship**: Liefert eine Liste von allen Hosted Services auf dem Gerät. Zu einem Hosted Service wird jeweils die Service ID (Endpoint) und der Servicetyp geliefert.

Folgendes Diagramm zeigt die beiden Metadata Exchange Nachrichten, welche über HTTP übertragen werden.

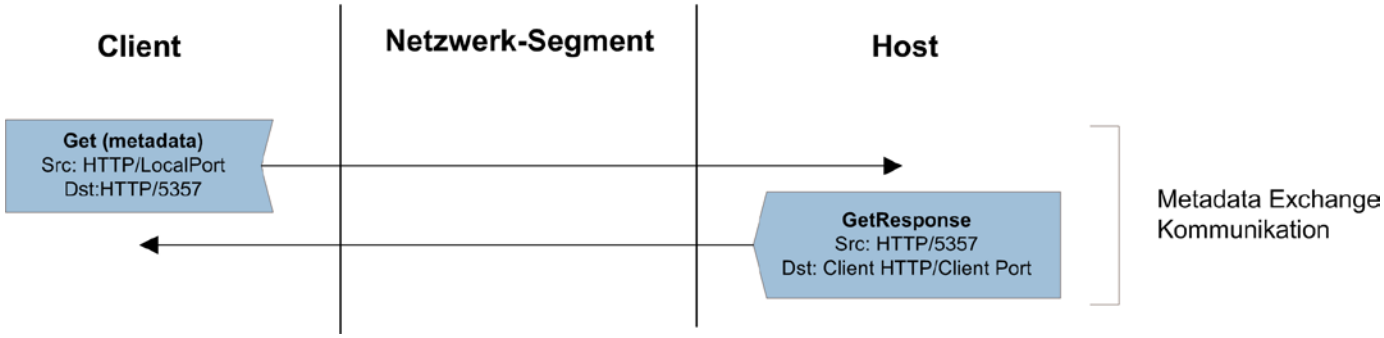


Abbildung 9: Netzwerkverkehr DPWS Metadata Exchange

Metadata Get Nachricht

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <soap:Header>
    <wsa:To>
      urn:uuid:bde0943a-0516-c8ca-80a6-000000b525ed
    </wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa:Action>
    <wsa:ReplyTo>
    <wsa:Address>
      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>urn:uuid:864528bb-045a-c8ca-a007-0000001463b9</wsa:MessageID>
  </soap:Header>
  <soap:Body>
  </soap:Body>
</soap:Envelope>
```

Metadata GetResponse Nachricht

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
  xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/discovery"
  xmlns:rco="http://schemas.hsr.ch/RoboterController">
  <soap:Header>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse</wsa:Action>
    <wsa:MessageID>urn:uuid:85fa1e49-045a-c8ca-b307-000000783a18</wsa:MessageID>
    <wsa:RelatesTo>urn:uuid:864528bb-045a-c8ca-a007-0000001463b9</wsa:RelatesTo>
    <wsd:AppSequence InstanceId="1196751381"
      SequenceId="urn:uuid:c883e4a8-9af4-4bf4-aaaf-06394151d6c0" MessageNumber="3"/>
  </soap:Header>
  <soap:Body>
    <wsx:Metadata>
      <wsx:MetadataSection
        Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel">
        <wsdp:ThisModel>
          <wsdp:Manufacturer>HSR</wsdp:Manufacturer>
          <wsdp:ManufacturerUrl>http://www.hsr.ch/</wsdp:ManufacturerUrl>
          <wsdp:ModelName>RoboterController</wsdp:ModelName>
          <wsdp:ModelNumber>12345</wsdp:ModelNumber>
          <wsdp:ModelUrl>http://www.hsr.ch/</wsdp:ModelUrl>
          <wsdp:PresentationUrl>http://www.hsr.ch/</wsdp:PresentationUrl>
          <wsdp:ModelName>MetadataProvidingModel</wsdp:ModelName>
        </wsdp:ThisModel>
      </wsx:MetadataSection>
      <wsx:MetadataSection
        Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice">
        <wsdp:ThisDevice>
          <wsdp:FriendlyName>HSR 2010 Roboter Controller</wsdp:FriendlyName>
          <wsdp:FirmwareVersion>demo</wsdp:FirmwareVersion>
          <wsdp:SerialNumber>12345678</wsdp:SerialNumber>
        </wsdp:ThisDevice>
      </wsx:MetadataSection>
      <wsx:MetadataSection
        Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship">
        <wsdp:Relationship
          Type="http://schemas.xmlsoap.org/ws/2006/02/devprof/host">
          <wsdp:Host>
            <wsa:EndpointReference>
              <wsa:Address>
                http://127.0.0.1:8084/cf16db78-02c9-c8ca-b37b-0000004071f6
              </wsa:Address>
            </wsa:EndpointReference>
            <wsdp:Types>rco:RoboterControllerServiceHost</wsdp:Types>
            <wsdp:ServiceId>urn:uuid:cf16db78-02c9-c8ca-b37b-0000004071f6</wsdp:ServiceId>
          </wsdp:Host>
          <wsdp:Hosted>
            <wsa:EndpointReference>
              <wsa:Address>http://127.0.0.1:8084/ec499d62-02c9-c8ca-b7ee-0000000bf3dd
              </wsa:Address>
            </wsa:EndpointReference>
            <wsdp:Types>rco:RoboterControllerService</wsdp:Types>
            <wsdp:ServiceId>urn:uuid:ec499d62-02c9-c8ca-b7ee-0000000bf3dd<wsdp:ServiceId>
          </wsdp:Hosted>
        </wsdp:Relationship>
      </wsx:MetadataSection>
    </wsx:Metadata>
  </soap:Body>
</soap:Envelope>
```

6.4 Eventing

Der DPWS Standard definiert einen Eventing Mechanismus, der es den Geräten erlaubt, die Clients bei einem bestimmten Ereignis zu benachrichtigen. Vom Prinzip ist es gleich wie die Events in .NET. Das Gerät stellt bestimmte Events zur Verfügung, bei denen sich interessierte Clients anmelden können. Event Benachrichtigungen sind einfache SOAP One-Way Nachrichten, welche asynchron behandelt werden.

Eine Anmeldung an einem bestimmten Event kann eine bestimmte Ablaufszeit haben. Der Client hat die Möglichkeit jede Anmeldung zu verlängern und den Status einer Anmeldung abzufragen. Eine Event Anmeldung kann auch ohne Ablaufszeit getätigt werden.

Folgende Nachrichten werden beim Eventing unterstützt:

- Subscribe
- Unsubscribe
- Renew
- GetStatus

6.4.1 Subscribe Nachricht

```
<s:Envelope …="">
  <s:Header …="">
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
    </wsa:Action>
    …
  </s:Header>
  <s:Body …="">
    <wse:Subscribe …="">
      <wse:EndTo>endpoint-reference</wse:EndTo> ?
      <wse:Delivery Mode="xs:anyURI"? >xs:any</wse:Delivery>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
      <wse:Filter Dialect="xs:anyURI"? > xs:any </wse:Filter> ?
      …
    </wse:Subscribe>
  </s:Body>
</s:Envelope>
```

6.4.2 Subscribe Response Nachricht

```
<s:Envelope …="">
  <s:Header …="">
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
    </wsa:Action>
    …
  </s:Header>
  <s:Body …="">
    <wse:SubscribeResponse …="">
      <wse:SubscriptionManager>
        wsa:EndpointReferenceType
      </wse:SubscriptionManager>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
      …
    </wse:SubscribeResponse>
  </s:Body>
</s:Envelope>
</s:Envelope>
```

6.4.3 Unsubscribe Nachricht

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
  <soap:Body>
    <Unsubscribe xmlns="http://schemas.microsoft.com/exchange/services/2006/messages">
      <SubscriptionId>e6fbf5c1-7e26-4bc6-a5f2-882063d5e34e</SubscriptionId>
    </Unsubscribe>
  </soap:Body>
</soap:Envelope>
```

6.4.4 Renew Nachricht

```
<s:Envelope …="">
  <s:Header …="">
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew
    </wsa:Action>
    …
  </s:Header>
  <s:Body …="">
    <wse:Renew …="">
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
      …
    </wse:Renew>
  </s:Body>
</s:Envelope>
```

6.4.5 GetStatus Nachricht

```
<s:Envelope …="">
  <s:Header …="">
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
    </wsa:Action>
    …
  </s:Header>
  <s:Body …="">
    <wse:GetStatus …="">
      …
    </wse:GetStatus>
  </s:Body>
</s:Envelope>
```

6.5 Analyse DPWS Implementierung

6.5.1 Vorhandene DPWS Implementierungen

Der DPWS Standard ist mittlerweile für mehrere Plattformen und in unterschiedlichen Programmiersprachen implementiert. Anbei eine Liste der bekanntesten DPWS Implementierungen:

Name	Beschreibung	Sprache	URL
WSDAPI	Microsofts native Implementierung des DPWS Standards (Web Services for Devices API). Vorhanden auf Windows Vista, Windows 7, Windows Embedded CE 6.0 R2 und Windows Server 2008	C/C++	*[URL1.2]*
.NET Micro Framework	Implementierung des DPWS Standards im .NET Micro Framework	C#	*[URL1.3]*
WS4D.org Java Multi Edition DPWS-Stack	Open Source DPWS Implementierung für Java Plattformen	Java	*[URL1.4]*
WS4D-gSOAP	Open Source DPWS Implementierung der Universität Rostock als Teil der WS4D Initiative	C/C++	*[URL1.5]*

Tabelle 3: Vorhandene DPWS Implementierungen

6.5.2 Umsetzungsentscheid

Die Aufgabenstellung schreibt zwar vor, dass als Resultat eine C# Compact Framework Implementierung des DPWS-Stacks vorhanden sein muss. Wie dieses Ziel erreicht wird, ist jedoch nicht vorgeschrieben. Konkret gibt es drei Varianten:

1. Managed Wrapper Interface über die Microsoft WSDAPI Implementierung
2. Portierung des DPWS Codes vom .NET Micro Framework
3. Neuimplementierung gemäss Standard

Die Entscheidung konnte anhand der verfügbaren Varianten schnell gefällt werden. Ein Wrapper Interface über die WSDAPI Implementierung kam nicht in Frage, da es kein WSDAPI für Windows Mobile gibt. Eine Neuimplementierung gemäss Standard hätte den Rahmen der Bachelorarbeit definitiv gesprengt. Man hätte sich zwar dafür entscheiden können den DPWS-Stack neu zu implementieren, dann hätte jedoch die Zeit für die Showcase Applikation nicht mehr gereicht.

Aus den genannten Gründen fiel die Entscheidung auf die Variante 2, welche eine Portierung des bestehenden Micro Framework Codes vorsieht. Rechtlich steht dieser Variante nichts im Wege, da der komplette Micro Framework Source Code unter der Apache 2.0 Lizenz veröffentlicht wurde. Diese Lizenz erlaubt eine kostenlose Weiterverwendung des Source Codes, ohne dass der Code der neu entstandenen Lösung offengelegt werden müsste. Des Weiteren bietet eine Portierung den Vorteil, dass Micro Framework Entwickler, welche den DPWS Code bereits kennen, sich sofort auch mit dem neuen Code im Compact Framework auskennen. Zusätzlich können bestehende Dokumentationen und Tutorials fürs Micro Framework ohne grosse Transferleistung verstanden werden.

Teil 4: Architektur & Design

7 Einführung Architektur & Design

Die folgende Abbildung 10 zeigt schematisch die Systemarchitektur der entwickelten Plattform. Darauf ist ersichtlich, wie der Remote Controller auf dem Mobilgerät mit dem MVC Pattern *[GAMMA95]* und der Roboter Controller auf dem Embedded System mit dem MVVM Pattern *[URL1.9]* entwickelt ist.

Die Kommunikation zwischen den beiden Geräten geschieht ausschliesslich gemäss dem DPWS Standard wobei als Übertragungsformat SOAP/HTTP bzw. SOAP/UDP zum Einsatz kommt.

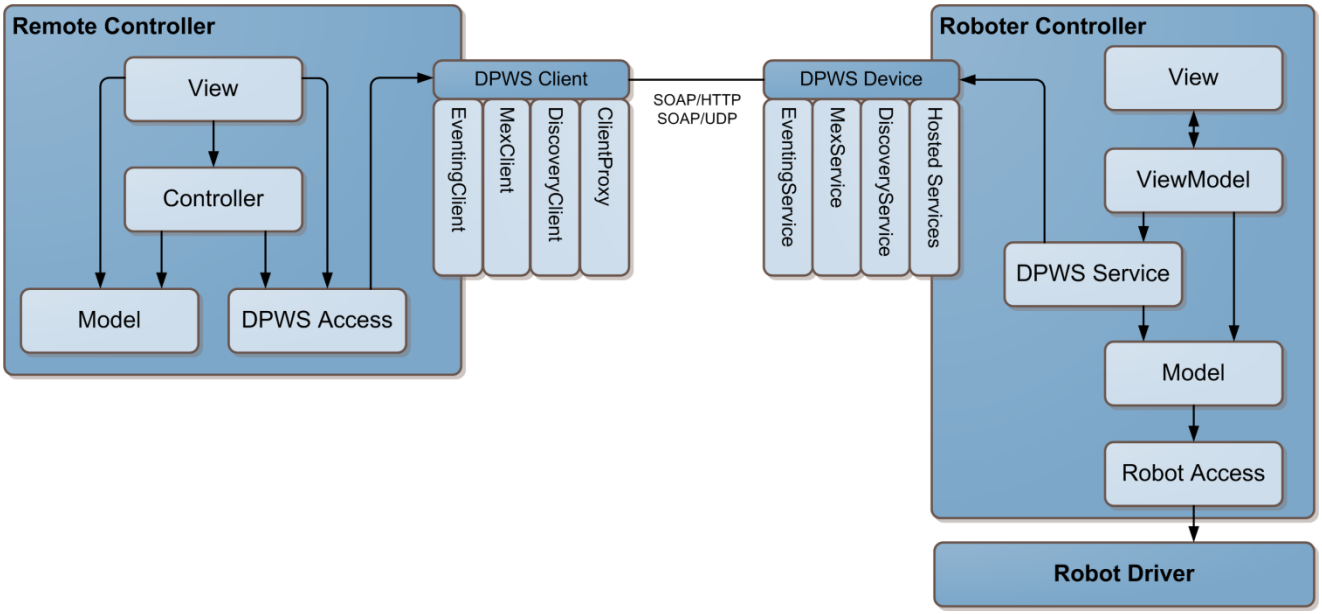


Abbildung 10: Übersicht Systemarchitektur

Das Design der drei Komponenten DPWS, Roboter Controller und Remote Controller werden in den folgenden Architekturkapiteln genauer erläutert und beschrieben.

8 Devices Profile for Web Services (DPWS)

Wie im Kapitel *6.5 Analyse DPWS Implementierung* bereits analysiert, wird für die DPWS Umsetzung der bestehende .NET Micro Framework Code aufs Compact Framework portiert. Der komplette Source Code kann mit dem Porting Kit auf der Microsoft Seite heruntergeladen werden *[URL1.7]*.

8.1 Portierung auf Compact Framework

Bei der Portierung des DPWS-Stacks aus dem .NET Micro Framework mussten einige Dinge angepasst werden. Das folgende Kapitel dokumentiert alle wichtigen Änderungen, welche für die Portierung auf das Compact Framework gemacht wurden.

8.1.1 Anpassungen am DPWS-Stack

Aus den Namespaces `System.Net` und `System.Net.Sockets` fehlten folgende Klassen:

InputNetworkStreamWriter, OutputNetworkStreamWriter, HttpListener, HttpListenerContext, HttpListenerRequest, HttpListenerResponse, NetworkStream.

Diese konnten alle auch aus dem .NET Micro Framework portiert werden.

Folgende Auflistung ist nicht komplett, sie soll einen Überblick über die gemachten Änderungen zur Portierung des DPWS Quellcode geben. Alle Änderungen sind im Quellcode kommentiert und können so nachvollzogen werden.

Thread.IsAlive
Im Compact Framework gibt es `Thread.IsAlive` nicht. Im Original Source Code wird dieses Property jedoch lediglich für unschöne Busy-Waits benutzt, welches im neuen DPWS-Stack mit einem `Thread.Join` optimiert wurde.

Angepasst in: WsUdpServer.cs:218, WsHttpServer:77

StreamWriter durch BinaryWriter ersetzt
Im Compact Framework gibt es keinen `StreamWriter`, welcher durch den vorhandenen `BinaryWriter` ersetzt wurde.

Angepasst in WsHttpServer.cs:602

Responsezugriff
Zur Laufzeit traten Probleme beim Zugriff auf die Response auf. Es stellte sich heraus, dass der Request explizit geschlossen werden muss, bevor auf die Response zugegriffen werden darf.

Änderungen in WsHttpClient.cs:121

Auslesen der IP
Bei dem portiertem DPWS-Stack kann ein Netzwerkinterface Anhand der IP ausgewählt werden. Beim alten wurde lediglich das erste Netzwerkinterface verwendet.

Änderungen in WsTransportUtilities.cs:245

XML Schreiben
Im Micro Framework fehlen einige Klassen zum Lesen und Schreiben von XML Dateien. Diese wurden speziell für das Micro Framework implementiert. Da diese im Compact Framework vollständig implementiert sind, wurden diese XML-Klassen komplett mit denen aus dem Compact Framework ersetzt.

Änderungen in diversen Klassen wo XML geschrieben wird.

Namespaces
Die Namespaces im Original Quellcode sind sehr unübersichtlich, da die Klassen nur grob aufgeteilt sind. Übersichtlichkeitshalber wurden diese Klassen in neue strukturiertere Namespaces aufgeteilt.

8.1.2 Assemblys

Der Output besteht aus mehreren Assemblys (Bibliotheksdateien):

Projekt/Output	Beschreibung
Dpws.Core.dll	Die Kernfunktionalität, welche vom Device und vom Client benutzt wird ist im Core enthalten.
Dpws.System.dll	Fehlende Klassen aus dem System.Net Namespace für das Compact Framework, welche aus dem Micro Framework portiert wurden.
Dpws.Device.dll	Enthält alle Klassen für die Implementation eines DPWS-Device.
Dpws.Client.dll	Enthält alle Klassen für die Implementation eines DPWS-Client.

Bei der Verwendung des DPWS-Stacks muss lediglich `Dpws.Device.dll` bzw. `Dpws.Client.dll` verwendet werden. Die anderen Assemblys werden intern für den DPWS-Stack benötigt und stellen gegen aussen keine Funktionalität zur Verfügung.

8.2 DPWS Codegenerierung

Für die Generierung von Proxies/Stubs und WSDL Dateien stellt das Micro Framework wie bei WCF ein Tool (`MFSvcUtil.exe`) zur Verfügung. Dieses Tool soll auch für den portierten

Code im Compact Framework vorhanden sein, weshalb auch dieser Code vom Micro Framework in ein neues Tool mit dem Name `Dpws.SvcUtil.exe` portiert wurde.

8.2.1 Anpassungen am Code

Bei der Portierung mussten ein paar wenige Anpassungen vorgenommen werden

- Jegliche Micro Framework Namespaces (`System.SPOT.*`) geändert
- Alte XML Klassen vom Micro Framework durch bestehende XML Klassen vom Compact Framework ersetzt
- Die XML Reader/Writer Klassen im Compact Framework haben einen kleinen Unterschied zu den Klassen im Micro Framework bezüglich dem Lese-Cursor während dem Parse Vorgang. So müssen alle `DataContractSerializer` von DPWS Event DataContracts geändert werden, dass sie in der `ReadObject()` Methode einmal mehr ein `reader.Read()` Aufruf tätigen. Ansonsten gibt es beim Versenden von Events stets eine `XmlException`.

```
public class AlarmOccurredResponseDataContractSerializer : DataContractSerializer
{
    public override object ReadObject(XmlReader reader)
    {
        AlarmOccurredResponse AlarmOccurredResponseField = null;
        if (reader.Read() && IsParentStartElement(reader, false, true))
        {
            ...
        }
    }
}
```

8.2.2 Erweiterungen am Code

Generiert man aus einem WSDL die dazu passenden Proxys/Stubs und Data Contract Serializer, so wird für den C# Namespace im Code stets der Target Namespace aus dem WSDL zu einem .NET Namespace konvertiert und so wiederverwendet. Der Target Namespace `http://schemas.hsr.ch/RobotControllerService` wird beispielsweise zu einem `schemas.hsr.ch.RobotControllerService`.

Da dies nicht immer dem gewünschten .NET Namespace entspricht, wurde dem Code Generator ein zusätzliches Kommandozeilenargument `/CodeNamespace:` bzw. `/NS:` erweitert, über das der Name des Namespaces im generierten Code explizit bestimmt werden kann.

8.3 Issues

8.3.1 Inkonsistenz Discovery Messages

DPWS ist eine Sammlung einiger WS-* Standards. Jedoch unterscheiden sich

Spezifikation	Discovery Message Value
DPWS 1.1	urn:docs-oasis-open-**org**:ws-dd:ns:discovery:2009:01
WS-Discovery	urn:docs-oasis-open:ws-dd:ns:discovery:2009:01

Die DPWS Spezifikation ist nur eine Sammlung von Standards und sollte aus diesen Gründen keine Inkonsistenz gegenüber diesen aufweisen. Aus diesem Grund wurde bei dieser Inkonsistenz so entschieden, dass man sich an den WS-Discovery Standard hält.

9 Roboter Controller (Device)

Unter der Komponenten Roboter Controller versteht man die ganze Applikation, welche auf dem Windows Embedded CE 6.0 Gerät läuft. Sie ermöglicht das Steuern des Roboters über ein Touchpanel und stellt zudem Web Services zur Verfügung, über welche der Roboter von einem mobilen Gerät gesteuert werden kann. Die Roboter Controller Applikation soll gemäss Anforderung fürs Compact Framework 3.5 in C# programmiert werden, wobei die View in Silverlight for Embedded (C++) umgesetzt werden soll.

9.1 Logische Architektur

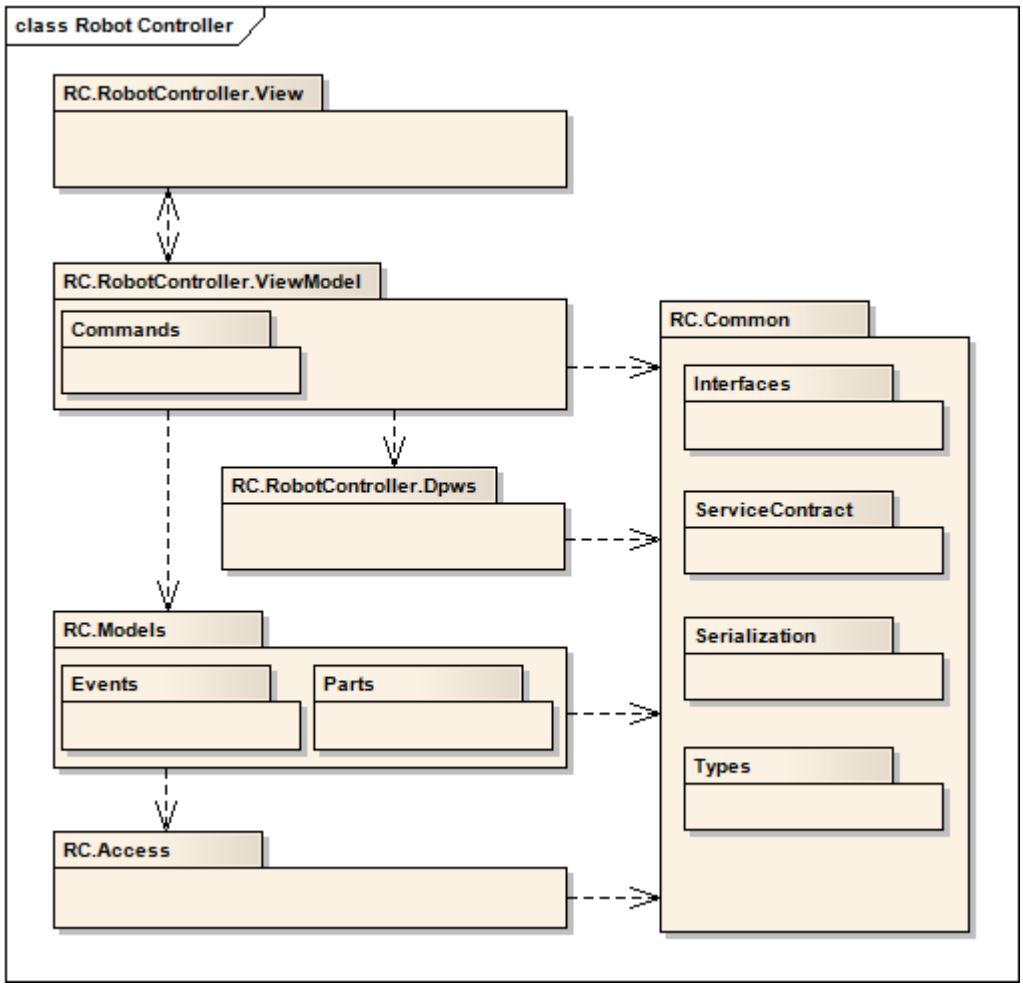


Abbildung 11: Logische Architektur Robot Controller

Die Logische Architektur in *Abbildung 11* zeigt die Abhängigkeit zwischen den Hauptpaketen in der Roboter Controller Applikation.

Hier fällt auf, dass zwischen View und ViewModel eine bidirektionale Abhängigkeit besteht. Dies ist bei dieser Architektur nicht zu umgehen, wenn man bedenkt, dass die View Komponente als native C++ DLL und das ViewModel in C# implementiert ist. So muss das ViewModel zwangsmässig wissen, wie View und deren exportierten Methoden heissen, um diese mittels P/Invoke Calls anzusprechen. Im Gegenzug muss die View für jeden nativen Callback einen Funktionspointer bereitstellen, welcher in der Signatur mit der Definition des managed Delegates übereinstimmt. Nichts desto trotz wurde bei der Implementierung darauf geachtet, dass die Abhängigkeit zwischen View und ViewModel so gering wie möglich gehalten werden. Weitere Details dazu sind im Kapitel *9.3.1 MVVM mit nativer View* dokumentiert.

9.2 Klassendiagramm

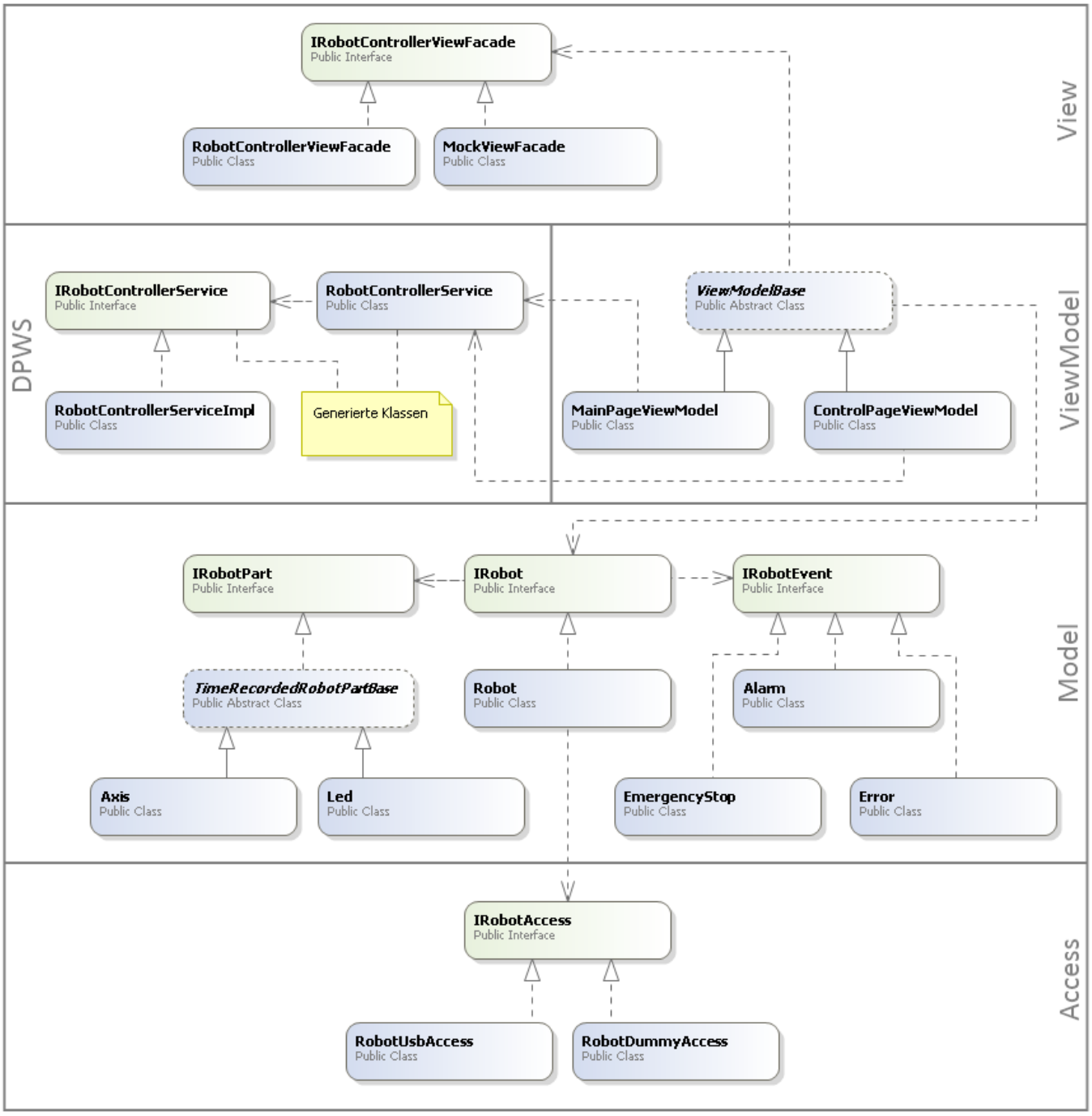


Abbildung 12: Klassendiagramm Roboter Controller

Das oben gezeigten Klassendiagramm visualisiert die wichtigsten Klassen der ganzen Roboter Controller Applikation. Die eingezeichneten Layer sind verteilt auf unterschiedliche Assemblys. Im Access Layer ist ersichtlich, wie für den Zugriff auf den Roboter zwei verschiedene Implementierungen des `IRobotAccess` Interface existieren. Solange der Roboter Treiber von Zühlke nicht vorhanden war, wurde stets die Klasse `RobotDummyAccess` verwendet, um den Zugriff auf den Roboter zu simulieren. Diese Klasse schrieb für jeden Steuerbefehl lediglich einen Eintrag in ein Log File (genauer beschrieben im Kapitel *9.5 Hardwareanbindung*).

9.3 Silverlight for Embedded

Das GUI der Roboter Controller Applikation wird mit Silverlight for Embedded umgesetzt, was zum ersten Mal mit dem Release von Windows Embedded CE 6.0 R3 (veröffentlicht im Oktober 2009) zur Verfügung stand. Silverlight for Embedded besitzt grundsätzlich über

die gleichen Features wie das gewöhnliche Silverlight 2, jedoch mit der Besonderheit, dass der Code-Behind in C++ geschrieben wird.

9.3.1 MVVM mit nativer View

Wie bereits erwähnt, wird der Roboter Controller fürs Compact Framework in C# geschrieben, wobei die View Komponente in C++ ist. Trotzdem soll ein Model-View-ViewModel Pattern zum Einsatz kommen, damit man die View unabhängig vom darunterliegenden Code im Expression Blend entwerfen kann. Das MVVM Pattern sieht im Roboter Controller folgendermassen aus:

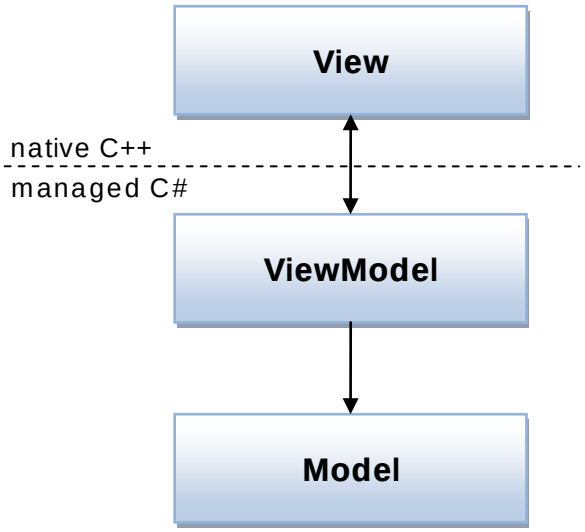


Abbildung 13: MVVM mit nativer View

Die Kommunikation vom ViewModel zur View geschieht über P/Invoke Calls aus dem C# Code heraus. Der umgekehrte Weg vom nativen C++ Code in den managed Code beruht darauf, dass man ein managed Delegate als nativer Function Pointer für Callbacks benutzen kann.

Managed → Native Kommunikation mittels P/Invoke

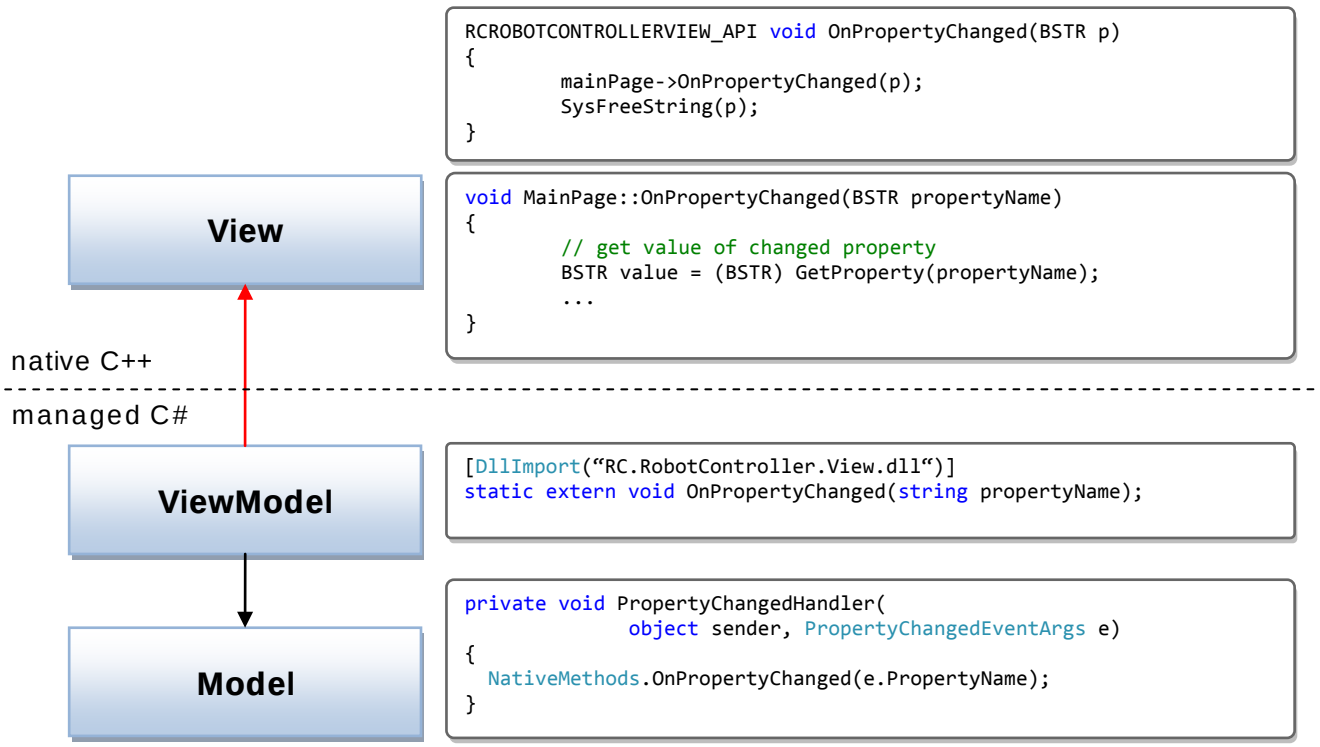


Abbildung 14: Kommunikation von Managed zu Native Code

Das Diagramm zeigt an einem Beispiel, wie das ViewModel die View benachrichtigt, wenn ein Property geändert wurde. Die Benachrichtigung geschieht über einen P/Invoke Call. Die View holt sich über ein generisches natives Callback den Wert des geänderten Properties und kann es so anschliessend darstellen.

Bereits hier ist ersichtlich, inwiefern das ViewModel die Methodensignatur der View kennen muss, um den P/Invoke Call zu tätigen.

Native → Managed Kommunikation mittels managed Delegates und nativen Callbacks
Damit die Kommunikation aus dem nativen Code zum managed Code funktionieren kann, muss zuerst ein managed Delegate im nativen Code als Funktionspointer registriert werden. Dazu ruft der managed Code initial mit P/Invoke eine native Methode auf, der er ein Delegate mitgeben kann, welches intern als Funktionspointer gespeichert wird. Die folgende Abbildung zeigt ein Beispiel, wie ein Delegate registriert wird.

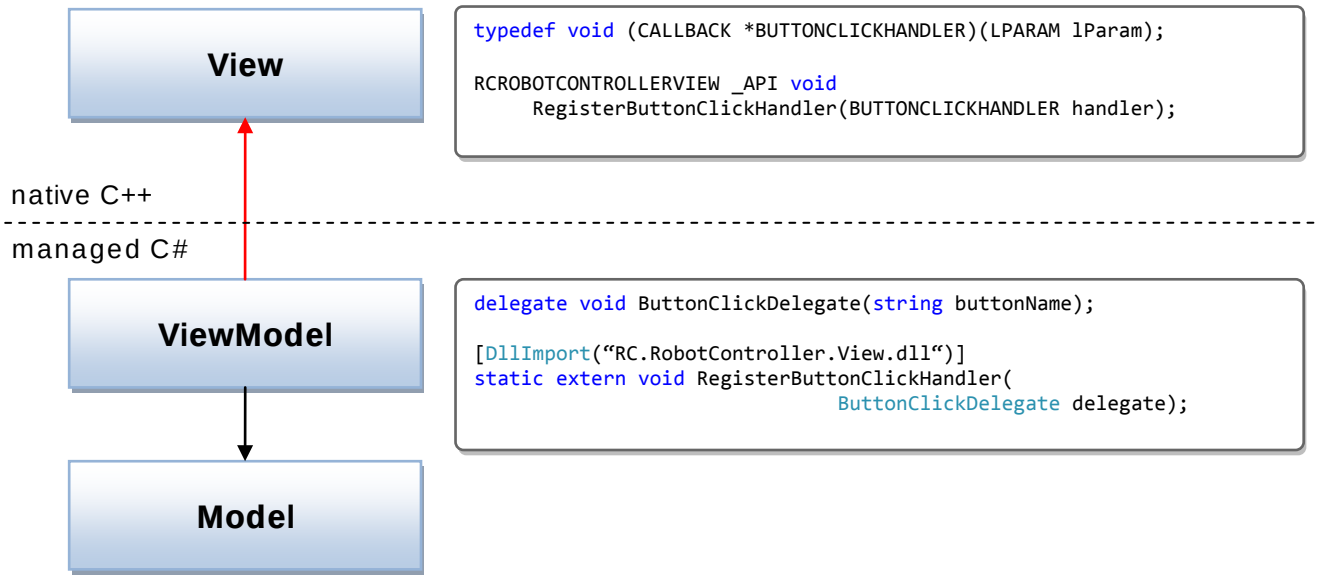


Abbildung 15: Registrieren eines Delegates als nativer Funktionspointer

Hier wird wieder ersichtlich, wie der native Funktionspointer (typedef) mit dem managed Delegate übereinstimmen muss, damit das Delegate im nativen Code registriert werden kann.

Nachdem nun das Delegate als Funktionspointer gespeichert ist, kann im C++ Code dieses Delegate jederzeit aufgerufen werden. Das folgende Beispiel zeigt, wie das managed Delegate aufgerufen wird, wenn ein Button gedrückt wurde.

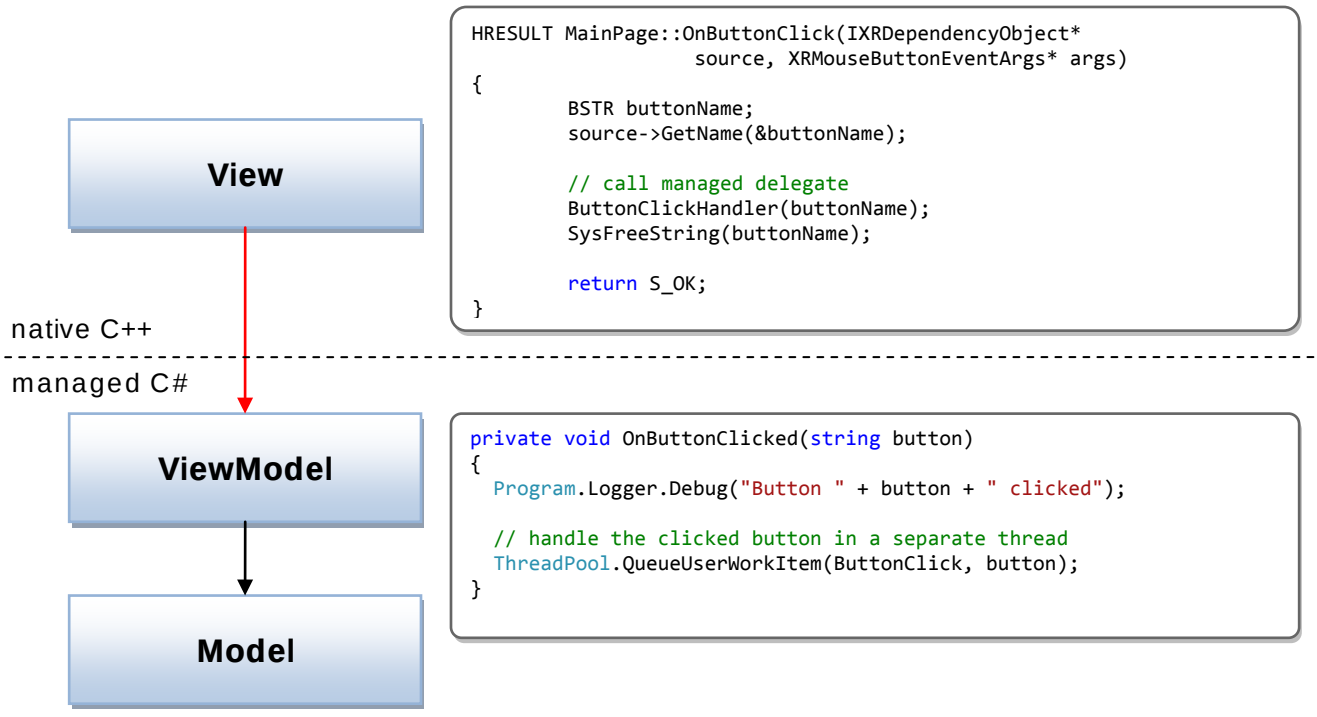


Abbildung 16: Kommunikation von Native zu Managed Code

9.3.2 Sequenzdiagramm Laden der nativen View

Das Sequenzdiagramm in *Abbildung 17* zeigt den Ablauf im Code, wenn die Applikation gestartet und native Silerlight View geladen wird. Die beiden Klassen auf der linken Seite sind in der ausführbaren Datei `RC.RobotController.exe` und komplett in C# geschrieben. Beim rechten Teil handelt sich um C++ Code für die View als DLL. Wie die Pfeile anzeigen, sind alle Methodenaufrufe von der `RobotControllerViewFacade` zum `RC.RoboterController.View` mittels P/Invoke implementiert. Der umgekehrte Weg von rechts nach links geschieht stets mit nativen Callbacks, welche zu Beginn vom managed Code registriert wurden.

Bevor die native View geladen wird, muss der managed Code ein Callback registrieren, welches vom nativen Code aufgerufen wird, sobald die View komplett geladen ist (`RegisterUiInitializedCallback()` Aufruf). Dies wird im managed Code benötigt, um weitere Callbacks registrieren zu können. Der `LoadMainPage()` Aufruf lädt anschliessend die Silverlight View. Sobald die View geladen wurde, meldet sie das Ereignis über den zuvor registrierten Callback, worauf der C# weitere Callbacks registrieren kann (z.B. Callbacks für Button-Click Ereignisse). Sobald nun ein Benutzer auf dem Silverlight GUI ein Button drückt, wird dieser Event sofort über den registrierten Callback dem C# Code gemeldet.

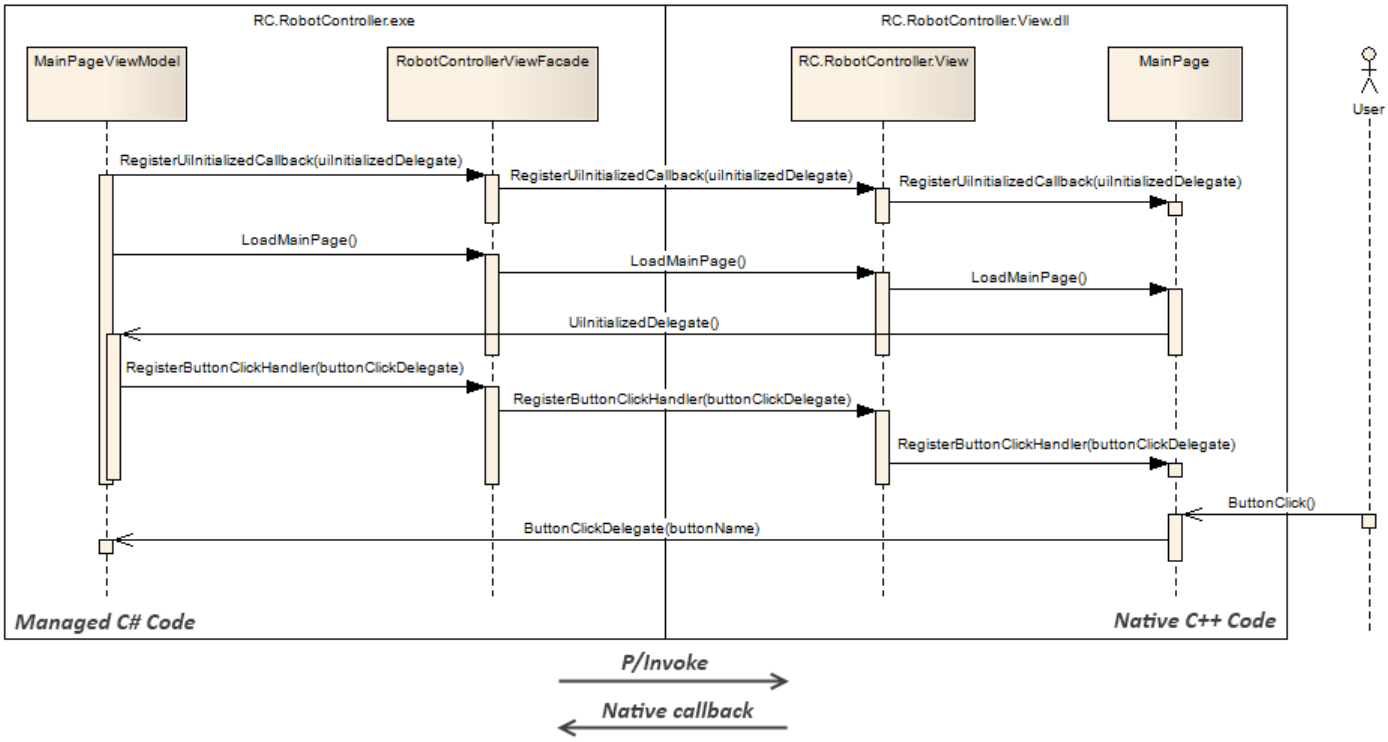


Abbildung 17: Sequenzdiagramm Laden der nativen View

9.3.3 Sequenzdiagramm Property Changed Mechanismus

Das nächste Sequenzdiagramm in *Abbildung 18* zeigt, wie die View über den Property-Changed Mechanismus benachrichtigt wird, wenn der Roboter ein Alarm meldet.

Das `MainPageViewModel` erhält über einen Event die Benachrichtigung, wenn ein Alarm auftritt. Dieses behandelt den Alarm, indem es die Alarmmeldung einem Property zuweist, welches beim setzen eines Wertes die `OnPropertyChanged()` Methode aufruft, welche wiederum zur `RobotControllViewFacade` weiterleitet, die anschliessend den P/Invoke Call zu nativen View tätigt. Die native View kann wiederum über ein Callback den Wert des geänderten Property vom ViewModel abfragen und anschliessen dem Benutzer darstellen.

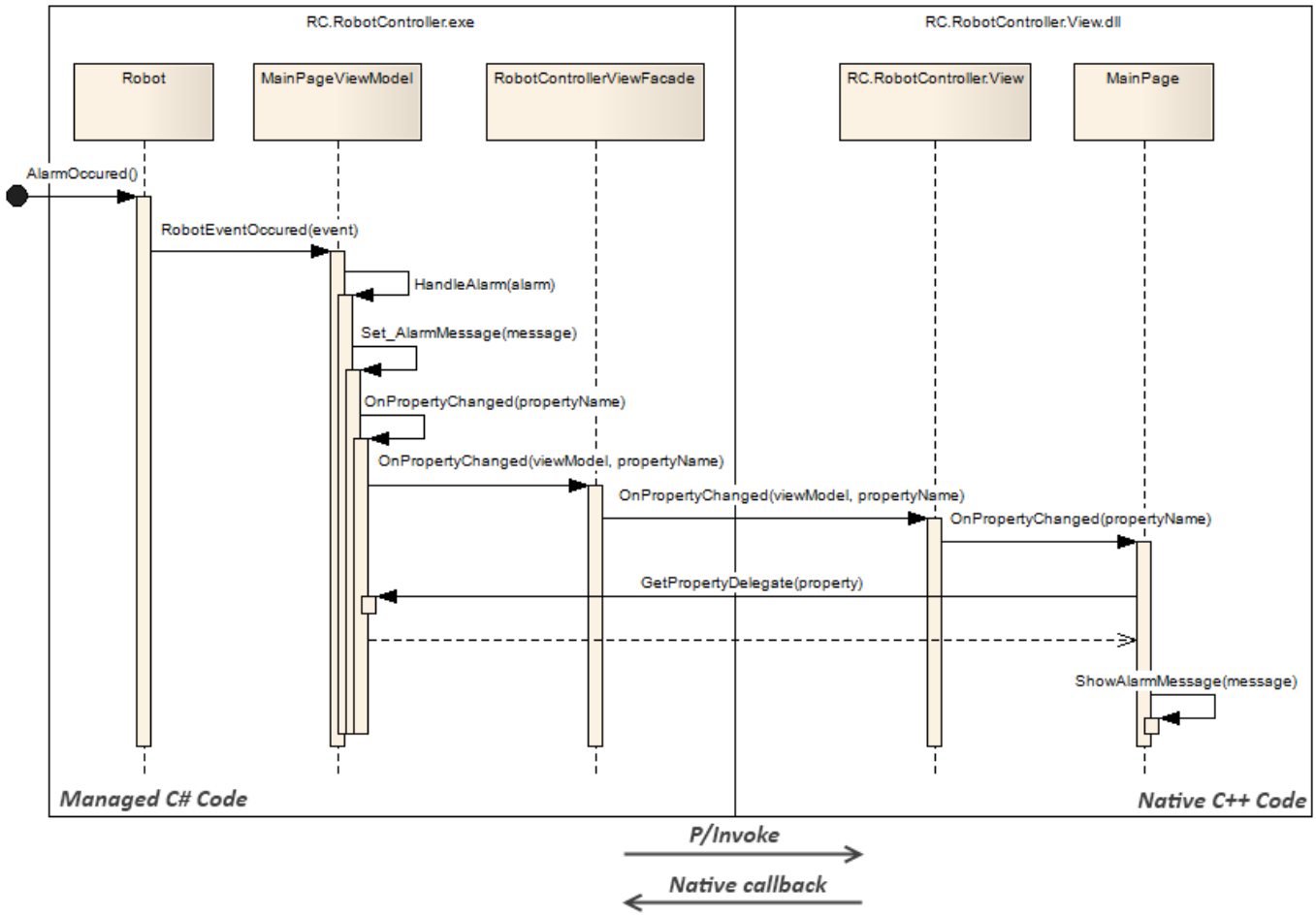


Abbildung 18: Sequenzdiagramm PropertyChanged Mechanismus

9.3.4 Tools für Silverlight for Embedded

Bei der Entwicklung des Prototyps wurde schnell festgestellt, dass Silverlight for Embedded sich noch im Anfangsstadium befindet. Es muss sehr viel Code geschrieben werden, der generiert werden könnte. So muss beispielsweise jedes GUI Element und jeder Eventhandler manuell im Code gebunden werden. Microsoft wird dies in zukünftigen Versionen verbessern.

Um die Entwicklung von Silverlight for Embedded Anwendungen effizienter zu gestalten, wurden in diesem Projekt zwei Tools eingesetzt.

XAML2CPP

Das Tool XAML2CPP wurde von Valter Minute (MVP für Windows Embedded) erstellt und dient der Generierung von C++ Code aus XAML Files. Das Tool ist frei verfügbar und kann auf der Website von Valter Minute heruntergeladen werden *[URL2.8]*. So kann die View wie gewohnt in Expression Blend entworfen werden, worauf anschliessend C++ Code generiert werden kann. Das Tool erstellt für eine XAML Datei eine entsprechende C++ Code-Behind Datei, in der bereits alle GUI Elemente und Eventhandler vorhanden sind.

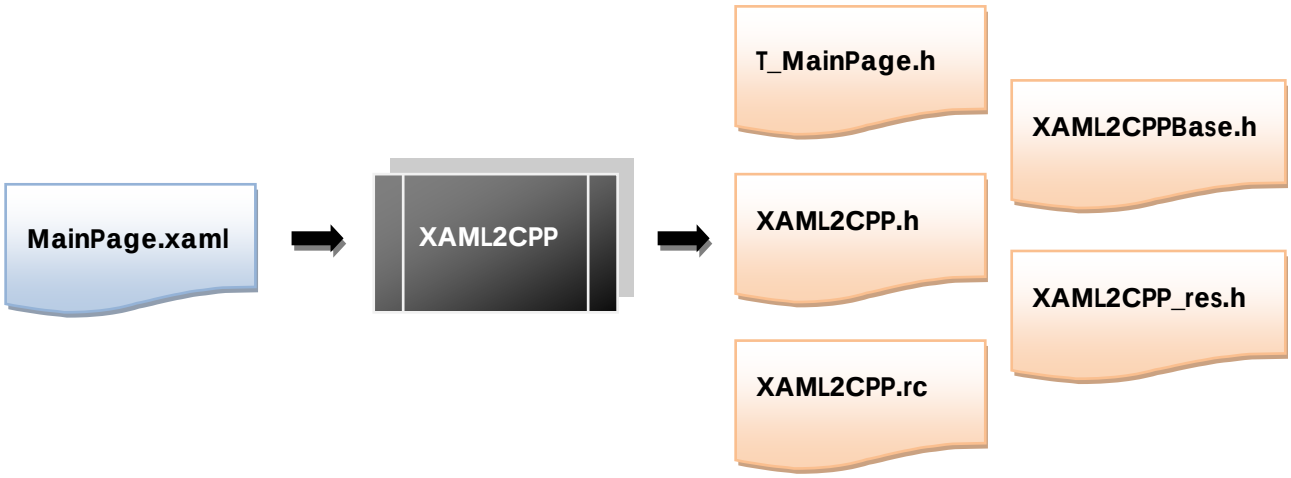


Abbildung 19: XAML2CPP Output

Aus einer XAML Datei generiert das XAML2CPP Tool folgende fünf Dateien:

Datei	Beschreibung
T_MainPage.h	TMainPage Klasse mit allen GUI Elementen und Bindings für Events
XAML2CPP.h	Header File welches jedes Header File aller XAML Views inkludiert
XAML2CPP.rc	Resource File für XAML Datei
XAML2CPPBase.h	Basis Klassen um Silverlight for Embedded zu starten
XAML2CPP_res.h	Typedefs für alle Resourcen (Resource identifier)

Tabelle 4: Beschreibung XAML2CPP Output

XamlCleaner

Das zweite Tool nennt sich XamlCleaner und wurde ihm Rahmen der Bachelorarbeit entwickelt um in Expression Blend 2 entworfene XAML Seiten so zu bereinigen, dass diese im C++ Projekt mitkompiliert werden können.

Ein Silverlight 2 Projekt muss eine gewisse Struktur aufweisen, damit dieses in Expression Blend 2 bearbeitet werden kann. Dass C++ Projekt (`RC.RobotController.View`) benötigt jedoch lediglich die XAML-Datei. Im Gegensatz zur „normalen“ .NET Silverlight Engine sucht die C++ Silverlight Runtime Ressourcen wie zum Beispiel Bilder nicht im Verzeichnis

der ausgeführten Datei, sondern verwendet als Basispfad „/“. Dies ist darauf zurückzuführen, dass die View als DLL geladen wird, und keine ausführbare Datei ist.

Der XamlCleaner automatisiert folgende Aufgaben:

1. Löschen der generierten Dateien (per Parameter konfigurierbar)
2. Kopieren aller XAML-Dateien vom Blend Projektverzeichnis ins C++ Projektverzeichnis
3. Relative Pfadangaben durch absolute ersetzen
4. Ausführen von XAML2CPP, damit C++ Dateien generiert werden

Dabei nimmt der XamlCleaner folgende Parameter entgegen:

Parameter	Beschreibung
XamlCleaner executable	Pfad zum XamlCleaner Programm
Expression Blend Project Path	Pfad zum Expression Blend Projekt
C++ Project Directory	Pfad zum C++ Projekt
Embedded Image Path	Absoluter Pfad zu den Bildern auf dem Embedded Device
Files to delete	Dateien, welche zuerst gelöscht werden. Best practice: alle generierten Dateien angeben.
Xaml2CPP Path	Pfad zum Xaml2CPP Programm

Tabelle 5: Parameter vom XamlCleaner

Der XamlCleaner sowie Xaml2CPP ist im TFS im Ordner `$(SolutionDir)Tools` eingecheckt, so dass ein Pre Build Event auf dem Projekt `RC.RobotController.View` konfiguriert werden kann, damit beim Kompilieren die XAML-Dateien automatisch kopiert und korrigiert werden:

```
$(SolutionDir)Tools\XamlCleaner.exe
$(SolutionDir)RC.RobotControllerViewDesign
$(SolutionDir)RC.RobotController.View
"\Program Files\RC.RobotController\images\\"
"XAML2CPP.h,XAML2CPP.rc,XAML2CPP_res.h,XAML2CPPBase.h"
$(SolutionDir)Tools\XAML2CPP.EXE
```

9.4 DPWS Device Implementierung

9.4.1 Sessionhandling

Gemäss Requirement 7 *„Kopplung/Pairing mit einem Client“* muss eine Kopplung zwischen Device und Remoteclient stattfinden, so dass jeweils nur ein Remotegerät gleichzeitig die Kontrolle über den RobotController besitzt.

Dies wurde auf Applikationsebene realisiert, indem bei einem erfolgreichem Verbindungsaufbau eine Session-GUID erstellt wird, welche bei allen Servicemethoden als Parameter mitgegeben werden muss. Die Methode `GetOperatingData` ist davon ausgeschlossen und kann ohne Session aufgerufen werden, so dass zum Beispiel von einer Management Zentrale aus die Betriebsdaten abgefragt werden können.

9.4.1 Robot-LAN

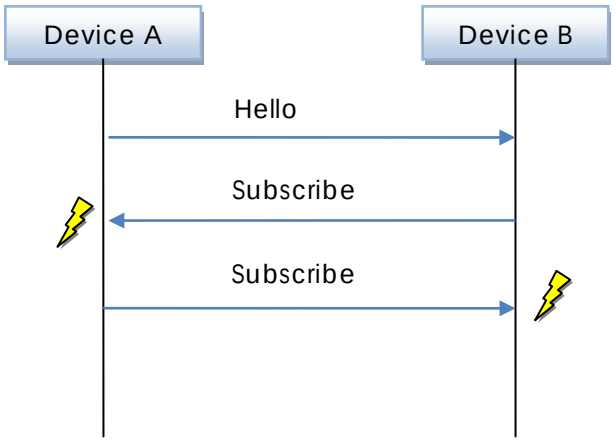
Sobald mehrere Roboter Controller in einem LAN zusammen verbunden sind, müssen diese miteinander kommunizieren können. Ein Notaus wird ins Robot-LAN gemeldet, so dass alle Roboter Controller in den Notauszustand gehen, bis das Notaus von einem beliebigen Robot Controller quittiert wird.

Um die Skalierbarkeit zu gewährleisten, müssten solche Meldungen für das Robot-LAN per Multicast oder Broadcast versendet werden. Beim DPWS ist Multicast jedoch nur für Dis-

covery Anfragen vorgesehen. Mit Eventing könnte ein solches Robot-LAN nachgebaut werden, indem sich die Roboter Controller gegenseitig registrieren. Dies würde wie folgt aussehen:

Szenario
Gerät B ist bereits „online“ während Gerät A eingeschaltet wird.

1. A sendet Hello Nachricht
2. B empfängt Hello Message und registriert sich bei A
3. Gerät A hat B noch nicht registriert und registriert sich bei B
4. Gerät B empfängt Registration, macht jedoch nichts, da B bereits A registriert hat.



Damit dies überhaupt möglich ist, muss die DPWS API um einen Event erweitert werden, so dass ein Gerät auf eine Registration (Subscribe) reagieren kann. Diese Erweiterung verändert nichts an der DPWS Spezifikation, sondern ermöglicht dem Entwickler auf Applikationsebene Logik zu implementieren.

Die Showcase Applikation soll den DPWS-Stack testen und demonstrieren. Des Weiteren werden für den Showcase nie grössere Roboternetze zum Einsatz kommen. Es ist und bleibt eine Demonstration resp. Simulation eines Roboternetzes. Eine „richtige“ Notaus Implementation müsste aus Performance- und Sicherheitsgründen direkt auf die elektrische Speisung gehen. Aus diesen Gründen haben wir uns entschieden, das Robot-LAN mit den vom DPWS zu Verfügung stehenden Mittel zu implementieren. Zugleich könnte so die Performance des DPWS Eventing getestet werden.

9.5 Hardwareanbindung

`IRobotAccess` ist die Schnittstelle zum Ansteuern eines Roboters. Es wurde ein Dummy-Treiber entwickelt (`RobotDummyAccess`), welcher alle Methodenaufrufe mit log4net loggt und auf der Konsole ausgibt, damit die Applikation auch ohne den Treiber von Zühlke getestet werden kann. Die Klasse `RobotUsbAccess` ist die eigentliche Implementation des USB-Treibers, welche die native Methodenaufrufe zum Treiber wrapped.

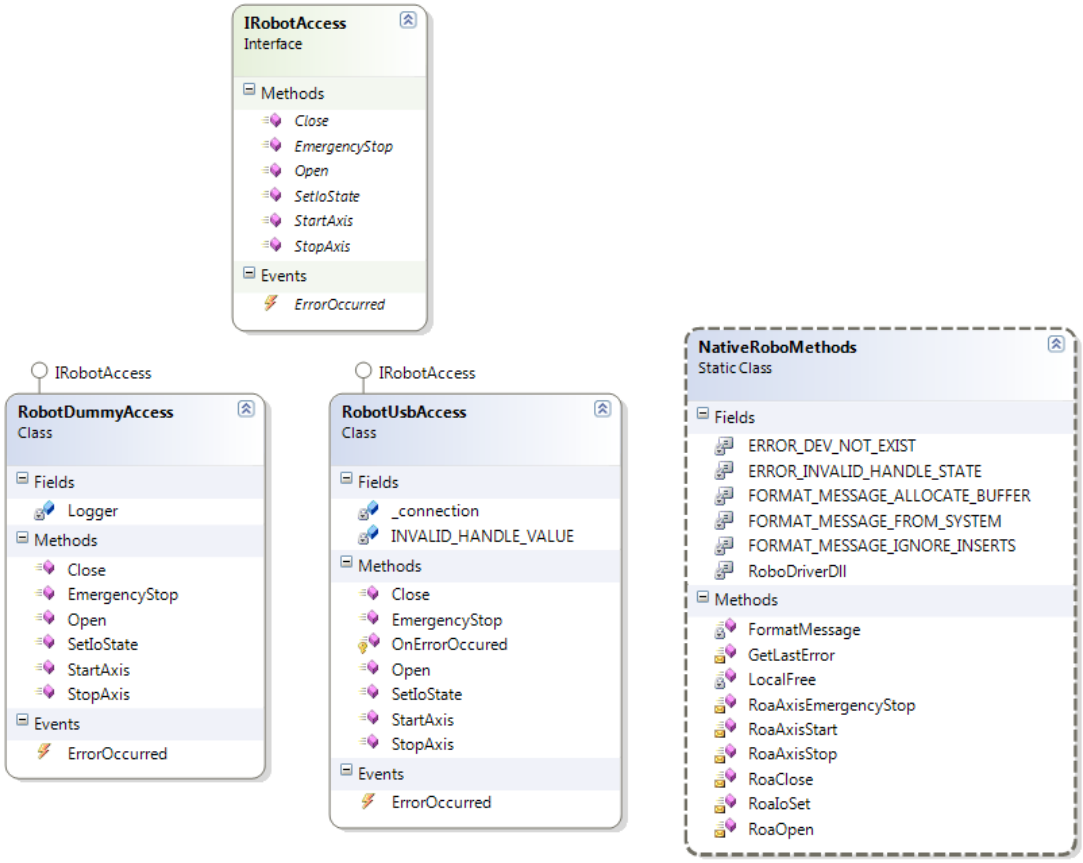


Abbildung 20: Implementation Roboter Treiber

Die Treiberimplementation ist unabhängig vom `RC.Models` Layer und benutzt darum eigene Enum Typen, welche direkt auf die nativen typedefs abgebildet werden. Zwischen den beiden Layers werden die Typen mit Integer Werten identifiziert. Dies hat den Vorteil, dass zum Beispiel das Hinzufügen einer neuen Achse im RC.Models keine kompilier oder Runtimeexceptions auslöst. Der Treiber erhält einen ungültigen Integer Wert (z.B. 6) welcher zu `RoboAxis.None` konvertiert wird. Beim Starten einer Achse mit dem Wert `RoboAxis.None` wird automatisch den `ErrorOccurred` Event gefeuert, welche eine entsprechende Meldung auf dem UI anzeigt.

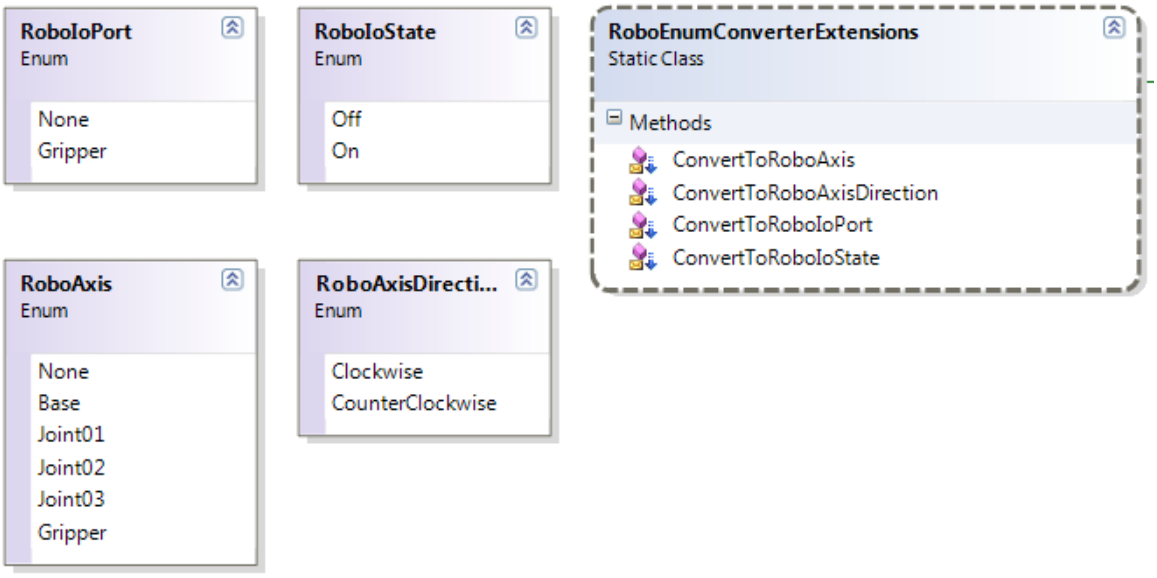


Abbildung 21: Treiber Enum Typen und deren converter extension methods

9.6 Konfigurationsdatei

Die Roboter Controller Applikation verfügt über eine Konfigurationsdatei `RC.RobotController.exe.config`, welche beim Start der Applikation geladen wird. Leider wird beim Compact Framework die Application Settings aus .NET nicht unterstützt. Die Struktur der XML Datei ist aus Kompatibilitätsgründen gleich aufgebaut. Diese Datei enthält `<appSettings>` für die Applikationseinstellungen und eine Konfiguration für den Log4Net Logger.

Der AppSettings Abschnitt enthält folgende Konfigurationen:

- **Key Mappings**: Jedem Steuerbefehl für den Roboter kann eine Taste der Tastatur zugewiesen werden. Dieser wird auf der Benutzeroberfläche entsprechend angezeigt.
- **Alarm Limits**: Konfigurierte Limits, wann für welche Achse ein Alarm ausgelöst werden soll.
- **Auto-Notaus Limit**: Limit für den automatischen Notaus
- **DPWS Konfiguration**: Informationen zum DPWS Device, welche über den Metadata Exchange den Clients bekannt gemacht werden

Der Log4Net Abschnitt enthält eine Logger Konfiguration für den normalen Logger, welcher alle auftretenden Fehler/Exceptions in ein Logfile schreibt. Eine zweite Logger Konfiguration ist für das Service Log eingetragen. Damit wird festgelegt in welche Datei das Service Log des Roboters geschrieben werden soll.

Die Konfigurationsdatei sieht folgendermassen aus:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <!-- Key Mapping -->
    <add key="StopKey"    value="Escape" />
    <add key="M1UpKey"    value="Down" />
    <add key="M2UpKey"    value="Q" />
    ... weitere Key Mappings

    <!-- Limits -->
    <add key="LimitM1"    value="00:01:00" />
    <add key="LimitM2"    value="00:01:00" />
    ... weitere Limits

    <!-- Auto Emergency Stop Limit -->
    <add key="LimitEmergencyStop" value="10" />

    <!-- DPWS Configuration -->
    <add key="DeviceEndpointAddress"        value="urn:uuid:BE028FBC-9934-4e1c-81B8-506795EA6579" />
    <add key="DeviceHostServiceNamespace"   value="http://schemas.hsr.ch/RobotControllerHost" />
    ... weitere DPWS Konfigurationen
  </appSettings>

  <log4net>
    <root>
      <level value="DEBUG" />
      <appender-ref ref="LogFileAppender" />
    </root>

    <appender name="LogFileAppender" type="log4net.Appender.RollingFileAppender" >
      <param name="File" value="log.txt" />
      <param name="AppendToFile" value="true" />
      <rollingStyle value="Composite" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="10MB" />
      <staticLogFileName value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <param name="ConversionPattern" value="%-7p%d{yyyy-MM-dd hh:mm:ss} - %m%n" />
      </layout>
    </appender>

    <appender name="ServiceLogAppender" type="log4net.Appender.RollingFileAppender" >
      <param name="File" value="servicelog.txt" />
      <param name="AppendToFile" value="true" />
      <rollingStyle value="Composite" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="100KB" />
      <staticLogFileName value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <param name="ConversionPattern" value="%d{yyyy-MM-dd hh:mm:ss} - %m%n" />
      </layout>
    </appender>
  </log4net>
</configuration>
```

10 Remote Controller (Client)

Die Client Komponente Remote Controller besteht aus einer WinForms Anwendung, die in C# fürs Compact Framework 3.5 entwickelt wurde. Die Komponente wird auf dem Windows Mobile 6.5 Gerät HTC HD2 betrieben.

10.1 Logische Architektur

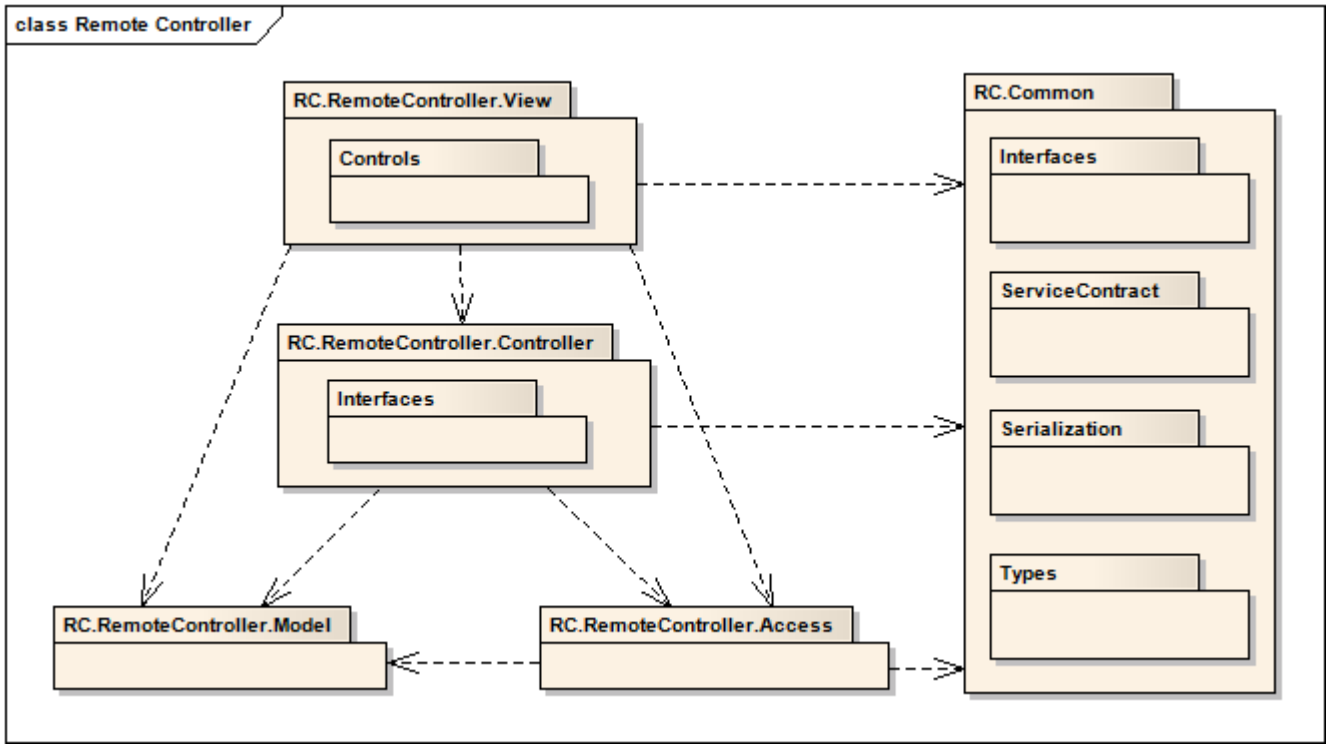


Abbildung 22: Logische Architektur Remote Controller

Die logische Architektur zeigt die Abhängigkeiten der wichtigsten Namespaces im Remote Controller. Wie auf der linken Seite der Abbildung ersichtlich ist, wurde die WinForms Applikation gemäss dem MVC Design Pattern entworfen.

10.2 Klassendiagramm

Im folgenden Klassendiagramm wird das eingesetzte MVC Pattern anhand zweier Views aufgezeigt. Die `MainForm` Klasse entspricht der ersten Ansicht in der Applikation, bei der nach vorhandenen Geräten gesucht werden kann. Verbindet man sich anschliessend zu einem bestimmten Gerät, gelangt man auf das `ControlForm`, worauf man den Roboter steuern kann.

Damit die Controller keine Abhängigkeiten zu den konkreten Views besitzen, schreiben sie für jede View ein Interface vor, welches die konkrete View implementieren muss (`IMainView` und `IControlView`). Jede View erstellt anschliessend seinen eigenen Controller und gibt über den Konstruktor eine Referenz auf sich selbst mit.

Wie es im MVC Pattern üblich ist, besitzen die View und der Controller jeweils eine Referenz auf die Model Klassen. Der Controller verändert aufgrund verschiedener Ereignisse die Werte im Model, worauf die View die veränderten Werte wiederum aus demselben Model liest und darstellt.

Das folgende Klassendiagramm ist nicht vollständig, übersichtlichkeitshalber werden nur die beiden komplexesten Views dargestellt.

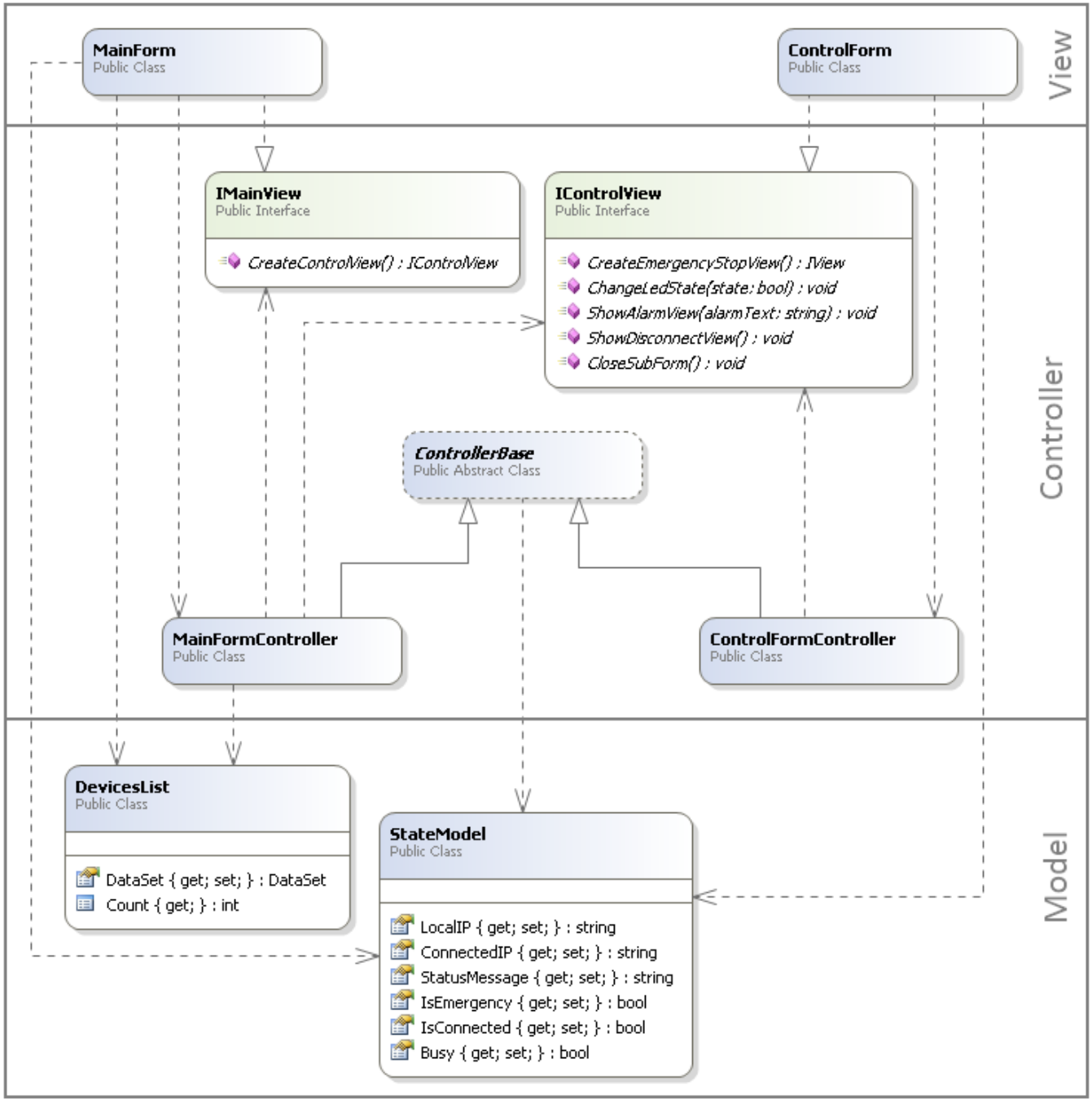


Abbildung 23: Klassendiagramm Remote Controller

10.3 Konfigurationsdatei

Wie der Roboter Controller besitzt der Remote Controller eine Konfigurationsdatei `RC.RemoteController.exe.config`, welche beim Starten der Anwendung geladen wird. Die Datei enthält alle Applikationseinstellungen unter `<appSettings>` und die Konfiguration für Log4Net.

Als Applikationseinstellung ist im Remote Controller lediglich ein Endpoint gespeichert. Diese eindeutige ID wird benötigt, um die einzelnen Mobilgeräte für die DPWS Kommunikation zu unterscheiden. Damit auch jedes Mobilgerät tatsächlich eine eindeutige ID erhält, ist initial kein Endpoint konfiguriert. Beim allerersten Start der Anwendung wird dann eine eindeutige ID (GUID) generiert und in der Konfigurationsdatei abgespeichert.

Unter `<log4net>` kann der Logger je nach Bedarf konfiguriert werden.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Endpoint" value="urn:uuid:BE028FBC-9934-4e1c-81B8-506795EA6579" />
  </appSettings>

  <log4net>
    <root>
      <level value="INFO" />
      <appender-ref ref="LogFileAppender" />
    </root>
    <appender name="LogFileAppender" type="log4net.Appender.RollingFileAppender" >
      <param name="File" value="log.txt" />
      <param name="AppendToFile" value="true" />
      <rollingStyle value="Composite" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="10MB" />
      <staticLogFileName value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <param name="ConversionPattern" value="%-7p%d{yyyy-MM-dd hh:mm:ss} - %m%n" />
      </layout>
    </appender>
  </log4net>
</configuration>
```

11 Visual Studio Struktur

Die *Abbildung 24* zeigt die im Visual Studio erstellte Projektstruktur. Nachfolgend werden die wichtigsten Projekte beschrieben.

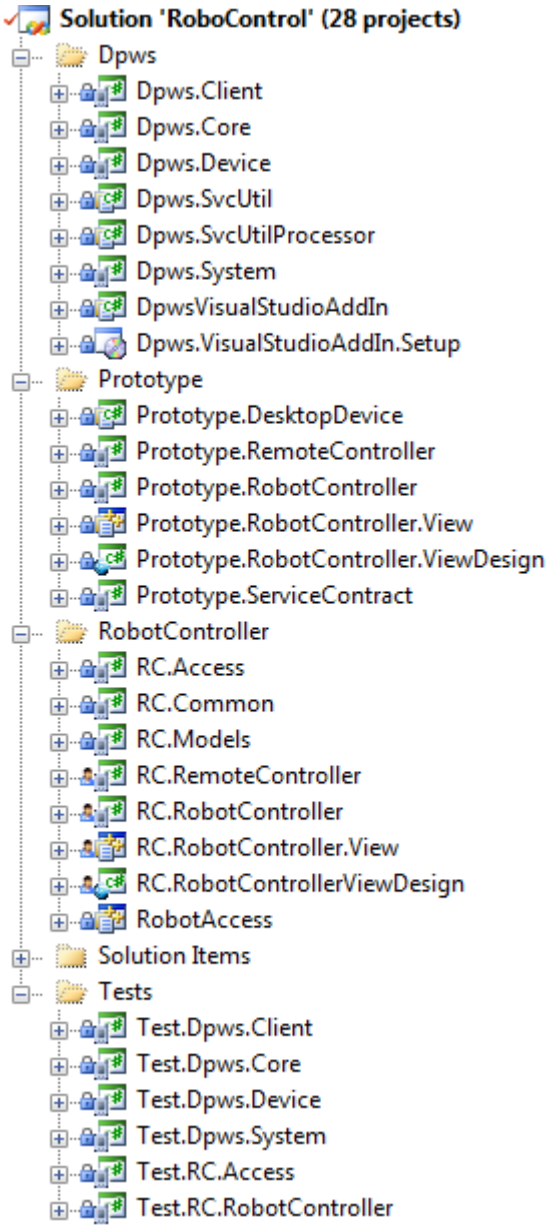


Abbildung 24: Visual Studio Struktur

Ordner Dpws

- **Dpws.Client:** Client Implementierung des DPWS Standards
- **Dpws.Core:** Kernfunktionalitäten des DPWS Standards, welche das Device und der Client benötigen
- **Dpws.Device:** Device Implementierung des DPWS Standards
- **Dpws.SvcUtil:** Kommandozeilen Tool zur Generierung von DPWS Code
- **Dpws.SvcUtilProcessor:** Logik der DPWS Codegenerierung
- **Dpws.System:** .NET Core Klassen, welche im Compact Framework nicht vorhanden sind und deshalb aus dem Micro Framework portiert wurden
- **DpwsVisualStudioAddIn:** Visual Studio Add-In Projekt, welches die Codegenerierung in die IDE integriert
- **Dpws.VsiualStudioAddIn.Setup:** Setup Projekt für das Visual Studio Add-In

Ordner Prototype

Enthält alle Projekte des Architekturprototyps, die in der Elaboration 2 Iteration erstellt wurden.

Ordner RobotController

- **RC.Access:** Access Layer der Roboter Controller Applikation für den Zugriff auf den Playtastic NC-1425 Roboter
- **RC.Common:** Enthält gemeinsam genutzte Klassen für Remote Controller und Roboter Controller. Neben den Service Contracts sind in diesem Assembly auch die Interfaces für jeden Layer im Namespace „Interfaces“ definiert.
- **RC.Models:** Model Layer der Roboter Controller Applikation. Beschreibt alle Model Klassen für den Playtastic NC-1425 Roboter.
- **RC.RemoteController:** Remote Controller Applikation, welche auf dem HTC HD2 Mobilegerät läuft.
- **RC.RobotController:** Teil der Roboter Controller Applikation, die im Compact Framework läuft. Dieses Assembly lädt zur Laufzeit die native Silverlight View.
- **RC.RobotController.View:** Silverlight for Embedded View Implementierung für die Roboter Controller Applikation
- **RC.RobotControllerViewDesign:** Separates Blend Projekt, welches zum Designen in Expression Blend erstellt wurde. Die darin erstellten XAML Dateien werden vom XamlCleaner automatisch ins RC.RobotController.View Projekt kopiert.
- **RobotAccess:** Enthält die C++ Header Files des Roboter Treibers von Zühlke. Dieses Projekt konnte verwendet werden um die P/Invoke Methoden zu definieren/testen.

Ordner Tests

Enthält alle Unit-Test Projekte der ganzen Solution

12 Deployment View

Abbildung 25 zeigt die Deployment View der Robot Controller Anwendung, verteilt auf die beiden Geräte Remote Controller und Robot Controller.

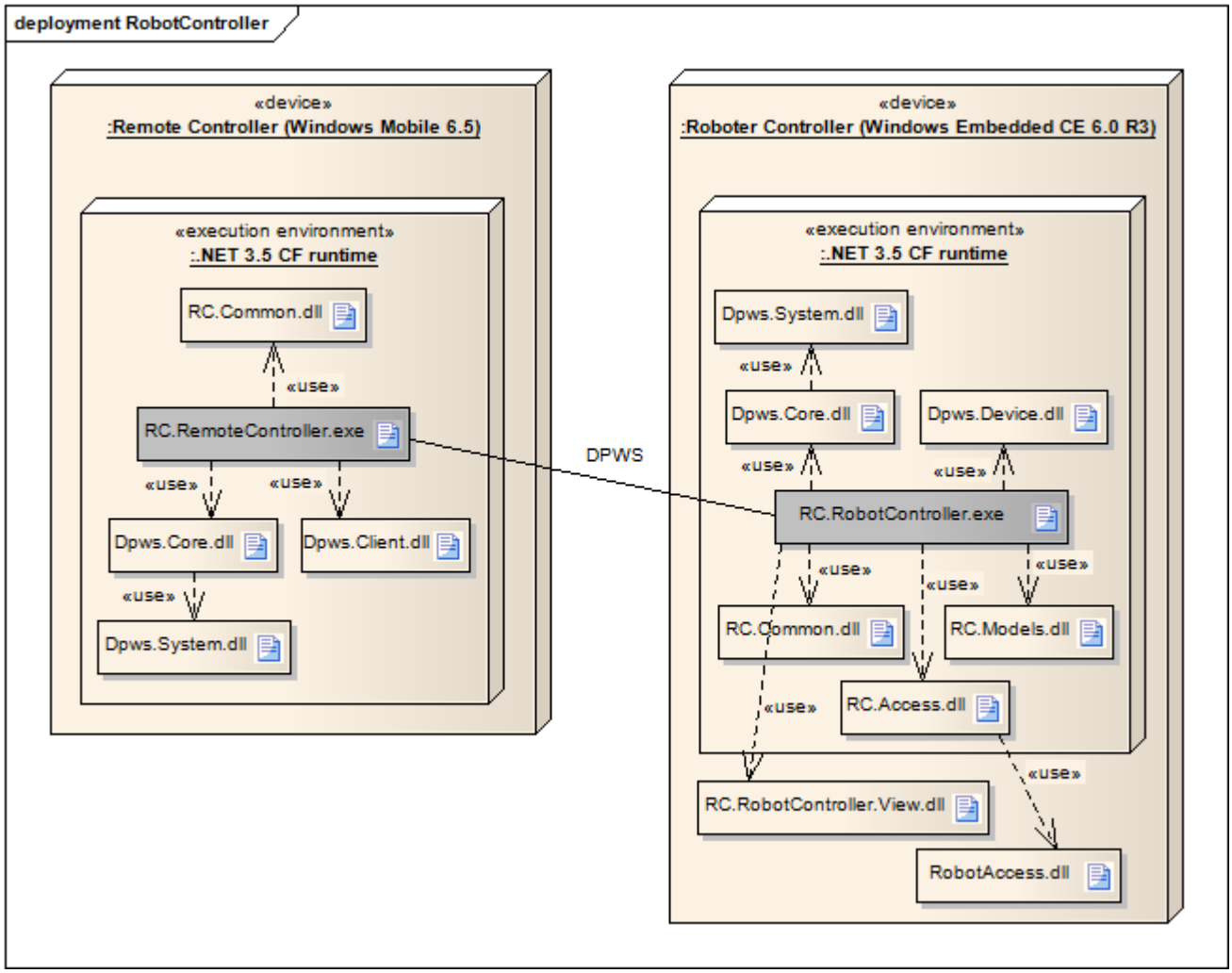


Abbildung 25: Deployment Diagramm

Die folgende Tabelle gibt einen Überblick über alle Artefakte:

Artefakt	Beschreibung	Zielsystem
RC.RobotController.exe	Ausführende Datei, welche die Steuerapplikation des Roboter Controllers startet	Roboter Controller
RC.RemoteController.exe	Ausführende Datei, welche die Steuerapplikation des Remote Controllers startet	Remote Controller
RC.RobotController.View.dll	Native Implementierung des UI in Silverlight for Embedded	Roboter Controller
RobotAccess.dll	Nativer Treiber um den Roboter zu steuern (implementiert von Zühlke)	Roboter Controller
RC.Common.dll	Gemeinsam genutzte Klassen wie z.B. Service Contracts, Interfaces, etc.	Roboter Controller Remote Controller
RC.Models.dll	Model Layer Implementierung des spezifischen Playtastic Roboter	Roboter Controller

RC.Access.dll	Access Layer Implementierung für den Zugriff/Steuerung des Playtastic Roboters	Roboter Controller
Dpws.Core.dll	Grundlegende DPWS Features, welche vom Remote und Roboter Controller gemeinsam genutzt werden	Roboter Controller Remote Controller
Dpws.Device.dll	DPWS Features für das Device	Roboter Controller
Dpws.Client.dll	DPWS Features für den Client	Remote Controller
Dpws.System.dll	HTTP/Net Klassen, welche im Compact Framework fehlen	Roboter Controller Remote Controller

Tabelle 6: Beschreibung der Artefakte

Teil 5: Ergebnisse & technische Erkenntnisse

13 Ergebnisse

Das Kapitel Ergebnisse fasst im ersten Teil alle erreichten Ziele zusammen. Es wird für jede der drei Komponenten kurz das Erreichte erläutert. Der zweite Teil widmet sich anschliessend den noch nicht komplett abgeschlossenen Anforderungen und beschreibt im Detail, was es noch zu tun gibt.

13.1 Erreichte Ziele

13.1.1 DPWS

Der komplette DPWS-Stack vom .NET Micro Framework wurde erfolgreich aufs .NET Compact Framework portiert, wobei Zühlke nun eine ausgereifte C# Implementierung des DPWS Standards fürs Compact Framework besitzt. Jegliche Funktionalität, welche im Micro Framework vorhanden war, gibt es nun auch fürs Compact Framework.

Code Generierung

Wie bei WCF das `SvcUtil.exe` gibt es im Micro Framework ein `MFSvcUtil.exe`, welches für die Generierung von Proxys/Stubs aus einer WSDL Datei oder einer WSDL Datei aus einem Assembly dient. Wie der ganze DPWS Code ist auch der Source Code des `MFSvcUtil` Tools als Open Source vorhanden. Mit dem Source Code als Basis wurde ein `Dpws.SvcUtil` Tool geschrieben, welches dieselbe Funktionalität besitzt wie das `MFSvcUtil`, ausser dass es Code fürs Compact Framework generiert anstelle des Micro Frameworks.

Das Code Generierungstool ist so als Command Line Tool entwickelt, dass es direkt in den MSBuild Prozess integriert werden kann.

Visual Studio Integration

Neben dem Tool um DPWS Code zu generieren, wurde ein einfaches Visual Studio Add-in entwickelt, welches das Dpws.SvcUtil Tool direkt im Visual Studio integriert. So können Entwickler direkt aus der Entwicklungsumgebung per Mausklick Service Contracts, Proxys/Stubs und WSDL Dateien generieren.

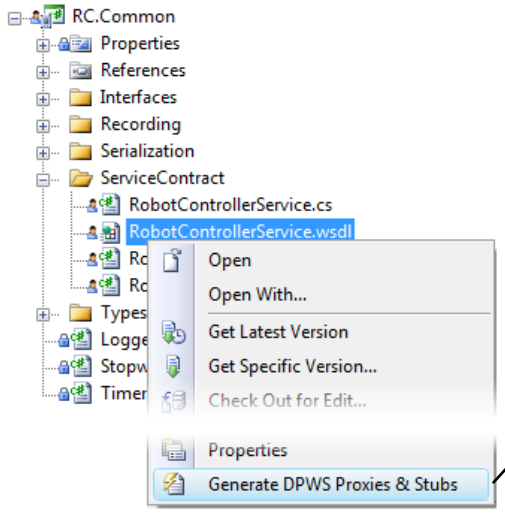


Abbildung 26: Visual Studio Add-in

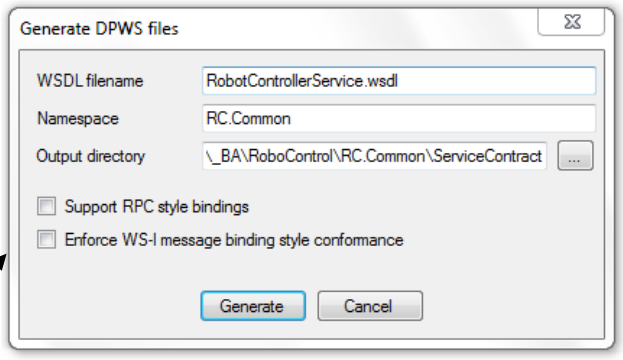


Abbildung 27: Visual Studio Add-in Dialog

13.1.2 Roboter Controller

Nachfolgend werden alle Hauptfunktionalitäten des Roboter Controllers erläutert. Wie gefordert wurde für die Entwicklung des UI Silverlight for Embedded eingesetzt.

Roboter steuern

Beim Starten der Applikation erscheint direkt die Ansicht um den Roboter zu steuern. Man hat wie gefordert die Möglichkeit den Roboter über das Touchpanel oder die Tastatur zu steuern. Bei jedem Button wird grau dargestellt, welche Taste der Tastatur der entsprechenden Funktion zugeordnet ist.

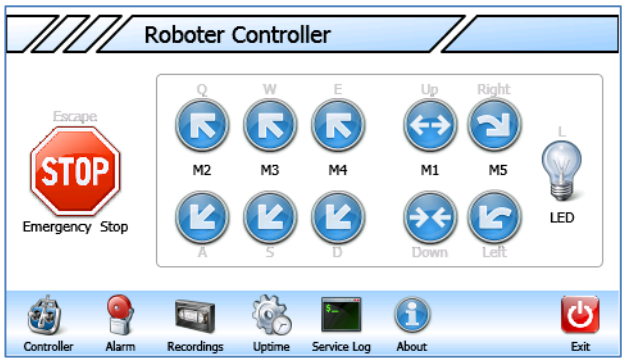


Abbildung 28: Roboter Controller Steuerung

Alarme konfigurieren

Die zweite Ansicht ermöglicht es dem Benutzer individuelle Alarmgrenzen für jede Achse und das LED zu setzen. Wird eine gesetzte Alarmgrenze überschritten, so wird automatisch ein Alarm ausgelöst und ein Service Log Eintrag geschrieben. Zusätzlich hat der Benutzer die Möglichkeit eine Zeitlimite zu setzen, bei der ein automatischer Notaus ausgelöst wird, wenn eine Achse ununterbrochen länger bewegt wird als die definierte Grenze.

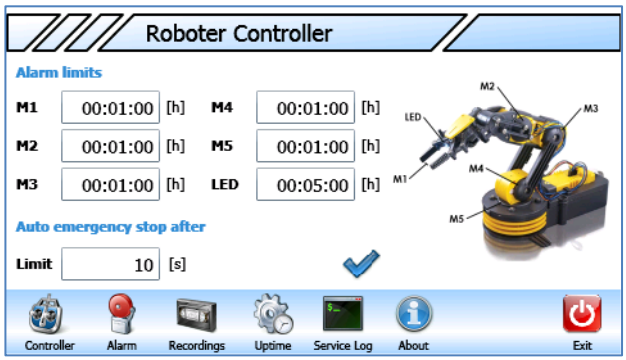


Abbildung 29: Alarm Konfiguration

Bewegungsabläufe

Wie gefordert, kann der Benutzer des Roboter Controllers beliebige Bewegungsabläufe aufzeichnen, speichern und anschliessend abspielen. Jeder Bewegungsablauf kann auch rückwärts abgespielt werden um den Roboter wieder in den vorhergehenden Zustand zu bringen.

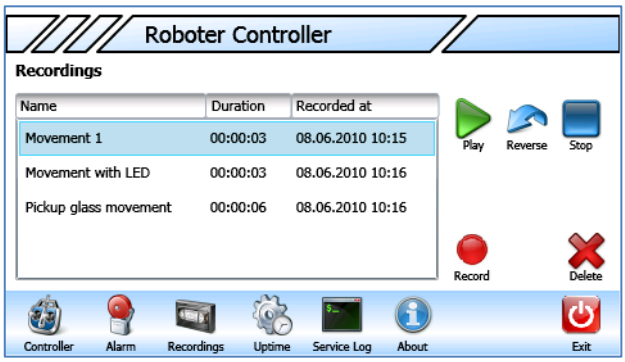


Abbildung 30: Bewegungsabläufe

Betriebsdaten abfragen

Die Betriebsdaten jeder Achse und der LED werden separat vom Roboter Controller kontinuierlich geführt und aktualisiert. Über eine separate Ansicht kann man sich die individuellen Betriebszeiten anzeigen lassen.

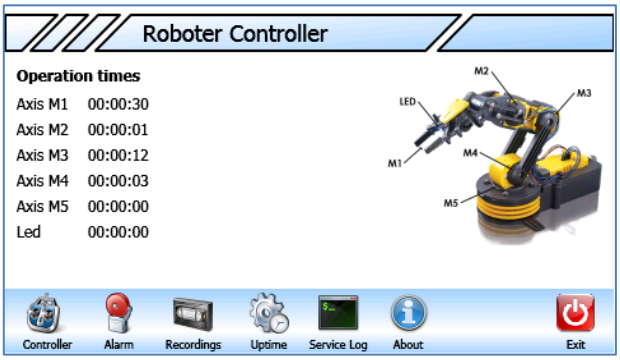


Abbildung 31: Betriebszeiten

Service Log

Jeder Fehlerzustand des Roboters, Alarm oder Notaus wird in einem separaten Service Log gespeichert. Dieses Log kann zu jederzeit auf dem Roboter Controller angeschaut werden.

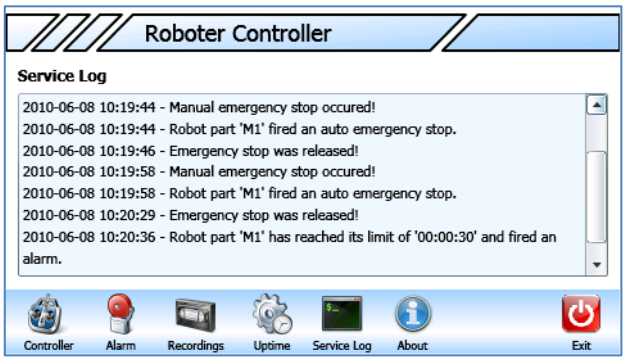


Abbildung 32: Service Log

Notaus

Wird die gesetzte Auto-Notaus Limite beim Bewegen einer Achse überschritten oder löst der Benutzer manuell ein Notaus aus, so werden alle Roboterbewegungen sofort angehalten und eine Meldung erscheint auf dem Roboter Controller. Solange diese Meldung aktiv ist, kann auf keine Art und Weise der Roboter weiter bedient werden.

Abbildung 33: Roboter Controller Notaus

Alarme

Überschreitet eine Achse oder LED eine gesetzte Alarmlimite, so wird ein Alarm ausgelöst, welcher auf dem Roboter Controller dargestellt und ins Service Log eingetragen wird. Bestätigt der Benutzer die Alarmmeldung, so wird der Alarm zurückgesetzt und solange deaktiviert, bis der Benutzer erneut einen Alarm konfiguriert.

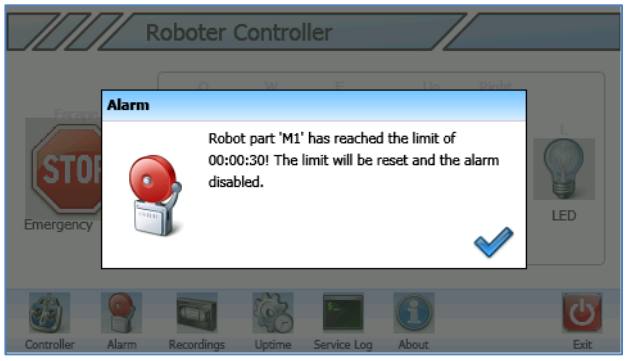


Abbildung 34: Roboter Controller Alarm

Kopplung zu Client

Verbindet sich ein Remote Controller mit dem Roboter Controller, so erscheint in der oberen rechten Ecke eine Meldung, dass ein Client verbunden ist. Ab diesem Zeitpunkt lässt der Roboter Controller keine weiteren Verbindungen von anderen Clients zu. Über den Disconnect Button kann die Verbindung zum verbundenen Client sofort aufgelöst werden.

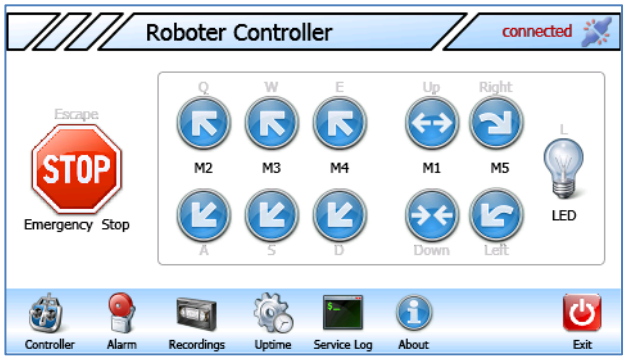


Abbildung 35: Roboter Controller Pairing

Roboternetz

Die Funktionalität des Roboternetzes ist auf dem GUI zwar nicht direkt ersichtlich, doch sobald ein weiterer Roboter Controller im LAN Segment gestartet wird, schliessen sich die Controller zu einem Roboternetz zusammen. Sie registrieren sich gegenseitig auf die Notaus Events. Sobald nun ein Roboter Controller in einen Notaus Zusand gerät, meldet er dies sofort allen weiteren Controllern im Netz weiter, welche anschliessend auch in den Notaus Zustand wechseln. Anschliessend kann auf einem beliebigen Roboter Controller das Notaus aufgelöst werden, worauf alle weiteren Controller wieder in den Betriebsmodus wechseln.

13.1.3 Remote Controller

Nachfolgend werden alle Hauptfunktionalitäten des Remote Controllers aufgelistet. Der Remote Controller wurde vollständig in C# fürs Compact Framework auf Windows Mobile 6.5 entwickelt.

Discovery

Beim Starten der Remote Controller Anwendung erhält man zuerst eine Ansicht um nach verfügbaren Roboter Controllern zu suchen. Dazu startet man den DPWS Discovery Prozess, worauf sich alle Geräte im Netz melden. Sobald der Discovery Prozess abgeschlossen ist, kann man eine Verbindung zu einem beliebigen Roboter Controller aufbauen.

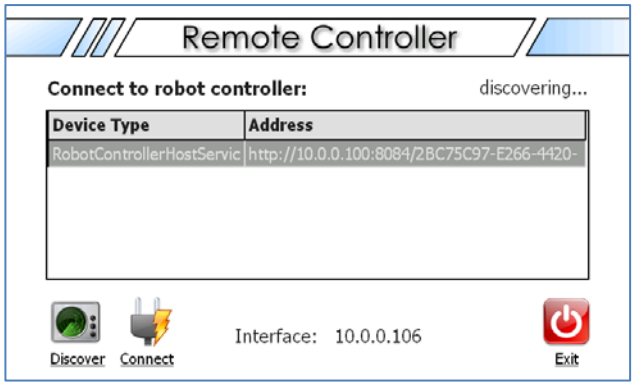


Abbildung 36: Remote Controller Steuerung

Roboter steuern

Wurde die Verbindung zum Roboter Controller erfolgreich hergestellt, erhält man auf dem Remote Client die gleiche Möglichkeit den Roboter zu steuern, wie direkt auf dem Touchpanel des Roboter Controllers. Man sieht die IP Adresse des verbundenen Controllers und kann die Verbindung jederzeit wieder abbauen.

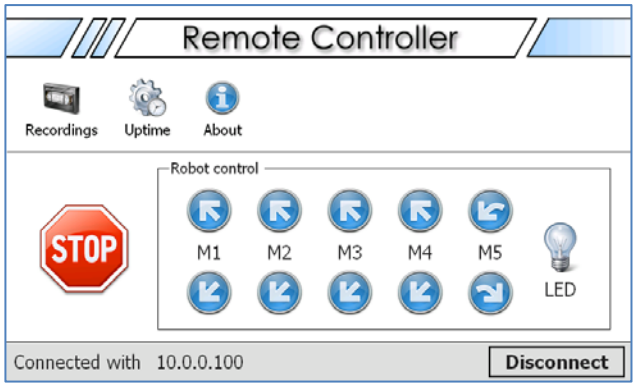


Abbildung 37: Remote Controller Steuerung

Bewegungsabläufe

Auf dem Remote Client können alle aufgezeichneten Bewegungsabläufe abgerufen und dargestellt werden. Somit können beliebige Abläufe auch vom Remote Client gestartet werden. Laufende Bewegungsabläufe können jederzeit gestoppt werden.

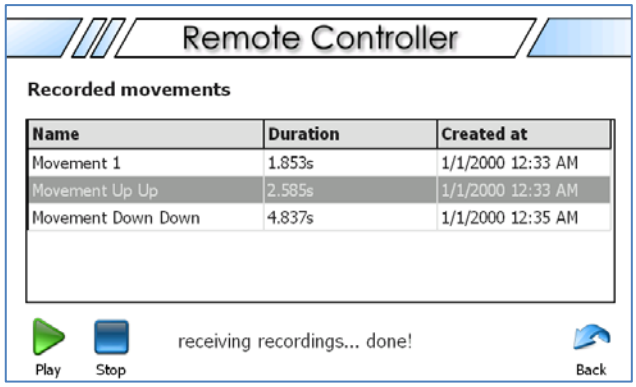


Abbildung 38: Bewegungsabläufe

Betriebsdaten abfragen

Wie auf dem Roboter Controller kann der Remote Client die Betriebszeiten jeder Achse und der LED abfragen.

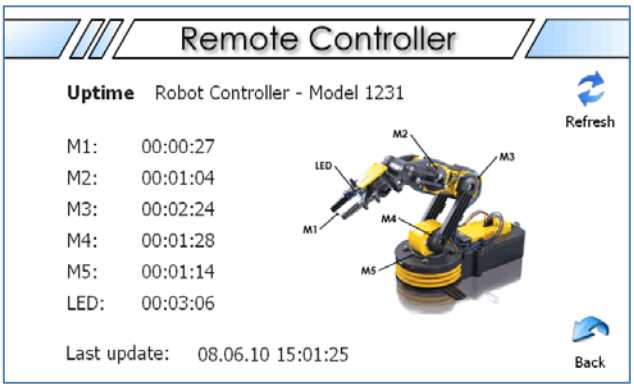


Abbildung 39: Remote Controller Betriebszeiten

Notaus

Wird auf irgendeinem Weg ein Notaus ausgelöst, so wird auf dem Remote Controller eine Meldung angezeigt und der Benutzer hat keine Möglichkeit mehr den Roboter zu steuern. Er muss warten, bis auf dem Roboter Controller das Notaus aufgelöst wird. Möchte der Benutzer nicht auf ein Auflösen des Notaus warten, so kann er lediglich die Verbindung zum Controller beenden, worauf er wieder auf die Discovery Ansicht gelangt.

Abbildung 40: Remote Controller Notaus

Alarme

Jegliche Alarm des Roboter Controller werden sofort dem verbundenen Client weitergeleitet und dort angezeigt.

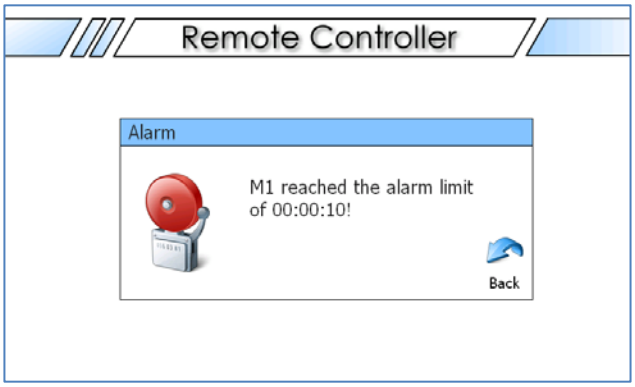


Abbildung 41: Remote Controller Alarm

Kopplung zu Device

Wenn ein Client eine Verbindung zu einem Roboter Controller hergestellt hat, so hat der Roboter Controller stets die Möglichkeit, die Kopplung zum Client zu beenden. Wird eine solche Kopplung manuell auf dem Roboter Controller unterbrochen, so wird dieses Ereignis dem Remote Controller dargestellt und er fällt anschliessend wieder auf den Discovery Screen zurück.

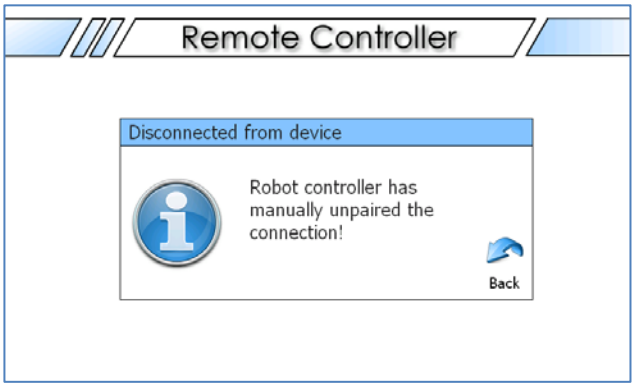


Abbildung 42: Remote Controller Kopplung aufgelöst

13.2 Offene Arbeiten

13.2.1 DPWS

R1.3 Komponente Description
Das Requirement R1.3 konnte mehrheitlich umgesetzt werden, bis auf die Unterstützung von WS-Policy. Die DPWS Implementierung auf dem Micro Framework unterstützt WS-Policy auch noch nicht und zeitlich lag es nicht mehr drin, dieser Teil komplett neu zu entwickeln.

R1.5 Komponente Security
Das Requirement R1.5 musste in Absprache mit dem Auftraggeber aus zeitlichen Gründen ganz weggelassen werden. Da die DPWS Implementierung auf dem Micro Framework den Security Teil auch nicht implementiert, war es terminlich nicht mehr möglich die Komponente komplett neu zu entwickeln.

IPv6 Unterstützung
Eine Unterstützung von IPv6 im DPWS-Stack war zwar nicht in der Anforderungsspezifikation vorgesehen, wird aber mit der zunehmenden Verbreitung von IPv6 sicher nützlich sein.

MTOM Test
Mit MTOM können Dateien verschickt werden. Dieser Teil wurde portiert, jedoch nicht mithilfe der Showcaseapplikation getestet.

13.2.2 Roboter Controller

R4.4 Roboter Controller agiert als Remote Controller
Das Requirement R4.4 musste in Absprache mit dem Auftraggeber aus zeitlichen Gründen ganz weggelassen werden. Es ist im Moment so gelöst, dass eine Meldung angezeigt wird, wenn der Controller keine Verbindung zum Roboter aufbauen konnte. Der Benutzer erhält dann die Möglichkeit nochmals zu versuchen eine Verbindung herzustellen oder den Roboter Controller zu beenden.

R7.1 Schutz nach Geräteklassen
Es wäre gemässe Requirement R7.1 gewünscht gewesen, dass der Roboter Controller nur auf Discovery Anfragen von gültigen Remote Controllern antwortet. Dieses Verhalten ist jedoch nicht gemäss DPWS Standard und sollte eher als Logik der jeweiligen Applikation umgesetzt werden. Ein solcher Schutz wurde in diesem Projekt nicht direkt umgesetzt. Es ist jedoch so gelöst, dass jeder Remote Controller zuerst eine Verbindung zum Roboter Controller aufbauen muss, worauf er eine Session-ID erhält. Alle folgenden Web Service Aufrufe zum Steuern des Roboters müssen mit einer gültigen Session-ID geschehen.

R11.4 Alarmlevels setzen
Als niedrig priorisiertes und optionales Feature wurde im Requirement R11.4 festgehalten, dass pro Motor verschiedene Alarmgrenzen definiert werden könnten. Aus zeitlichen Gründen wurde diese Anforderung nicht mehr umgesetzt und dafür mehr Zeit in wichtigere Features investiert.

Roboternetz Darstellung
Bei der Entwicklung des Roboternetzes ist aufgefallen, dass es nützlich wäre, wenn der Roboter Controller eine Ansicht liefern würde, bei der ersichtlich ist, welche Roboter im aktuellen Roboternetz vorhanden sind. Dies war keine explizite Anfoderung und wurde deshalb aus Zeitgründen nicht mehr umgesetzt.

13.2.3 Remote Controller

ImageButton erhält teilweise kein ButtonUp Event
In den Systemtests ist auf dem Remote Controller ein Problem mit dem eigenen `ImageButton` Control aufgefallen. Teilweise kann es vorkommen, dass das Control kein `ButtonUp` Event erhält, wenn der Benutzer den Button nur sehr kurz antippt. Tritt dieser Fall ein, so startet eine Roboterbewegung und wird nicht mehr beendet. Um die Bewegung zu stoppen muss man anschliessend denselben Button nochmals drücken oder alle Bewegungen per Notaus anhalten.

14 Technische Erkenntnisse

Im folgenden Kapitel wird auf Probleme bzw. Erkenntnisse hingewiesen, die im Verlaufe der Bachelorarbeit aufgetreten sind. Zu jedem Punkt wird erläutert, wie das Problem gelöst werden konnte.

14.1 Windows Embedded Device

14.1.1 Deployment auf Embedded Device

Problem
Wie wird eine Applikation aufs Embedded Device deployed? Wie wird eine Verbindung zum Embedded Device hergestellt?

Lösung
Um aus dem Visual Studio eigene Applikationen aufs Embedded Device zu deployen werden die CoreCon Utilities benötigt. Falls das Device über einen USB Anschluss verfügt, über den man mittels ActivSync aufs Device verbinden kann, so werden die CoreCon Utilities automatisch aufs Gerät kopiert und das Deployment kann ohne weitere Konfiguration vorgenommen werden. Ist dies nicht der Fall, so müssen die CoreCon Dateien entweder direkt im OSDesign des Embedded Betriebssystem integriert werden oder diese manuell aufs Gerät kopiert werden. Die CoreCon Utilities sind auf dem Windows CE 6.0 Entwicklungsrechner im Ordner „`C:\Program Files\Common Files\Microsoft Shared\CoreCon\1.0`“ zu finden.

Wenn nun keine Verbindung über ActiveSync zum Gerät hergestellt werden kann, so müssen auf dem Embedded Gerät manuell die CoreCon Anwendungen `conmanclient2.exe` und `cmaccept.exe` ausgeführt werden, um anschliessend via WLAN/Ethernet über TCP/IP aufs Gerät verbinden zu können.

14.1.2 Kein Senden oder Empfang von TCP/UDP Paketen bei ActiveSync Verbindung

Problem
Embedded Gerät empfängt/versendet keine TCP/UDP Pakete, solange über USB eine ActiveSync Verbindung zum Entwickler-PC hergestellt ist.

Lösung
Die Konfiguration muss an zwei Orten vorgenommen werden:

Beim Einstecken des Telefons erscheint das Windows Mobile Device Center. Falls nicht, kann das Windows Mobile Device Center im Tray Menü gestartet werden. Dort unter Mobile Device Settings → Connection Settings muss unter “This computer is connected to:” auf Work Network (Deutsch: Firmennetz) ausgewählt werden. Bei „The Internet“ wird automatisch der Netzwerkadapter ausgewählt, welcher eine Verbindung ins Internet machen kann. Dies ist gerade bei Testnetzwerken die falsche Wahl.

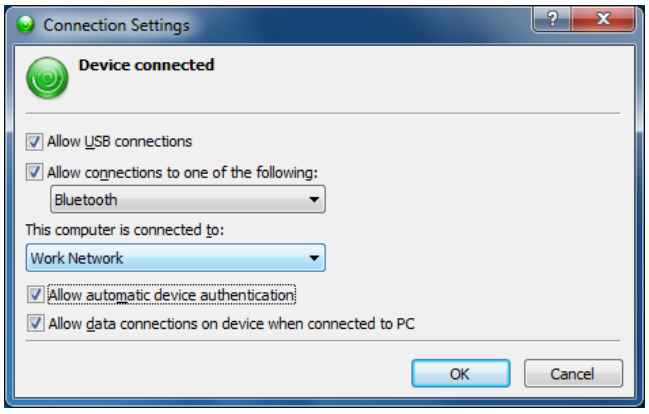


Abbildung 43: Windows Mobile Device Center Einstellungen

Desweiteren muss auf dem Telefon selber auch noch eine ähnliche Konfiguration getätigt werden.

Navigation zu den Verbindungseinstellungen:

HTC Sense:
Einstellungen (Home-Slider) → Funkeinstellungen → Menü → Verbindungen

Windows Mobile 6.5:
Start -> Einstellungen -> Verbindungen -> Verbindungen

Im folgenden Menü auf Erweitert → Netzwerke auswählen. Dort müssen beide Drop-Down Menüs auf „Firmennetzwerk“ wie auf der Abbildung ersichtlich konfiguriert sein.

14.2 Windows Mobile/Embedded Emulator

14.2.1 UDP Multicast Empfang nicht möglich

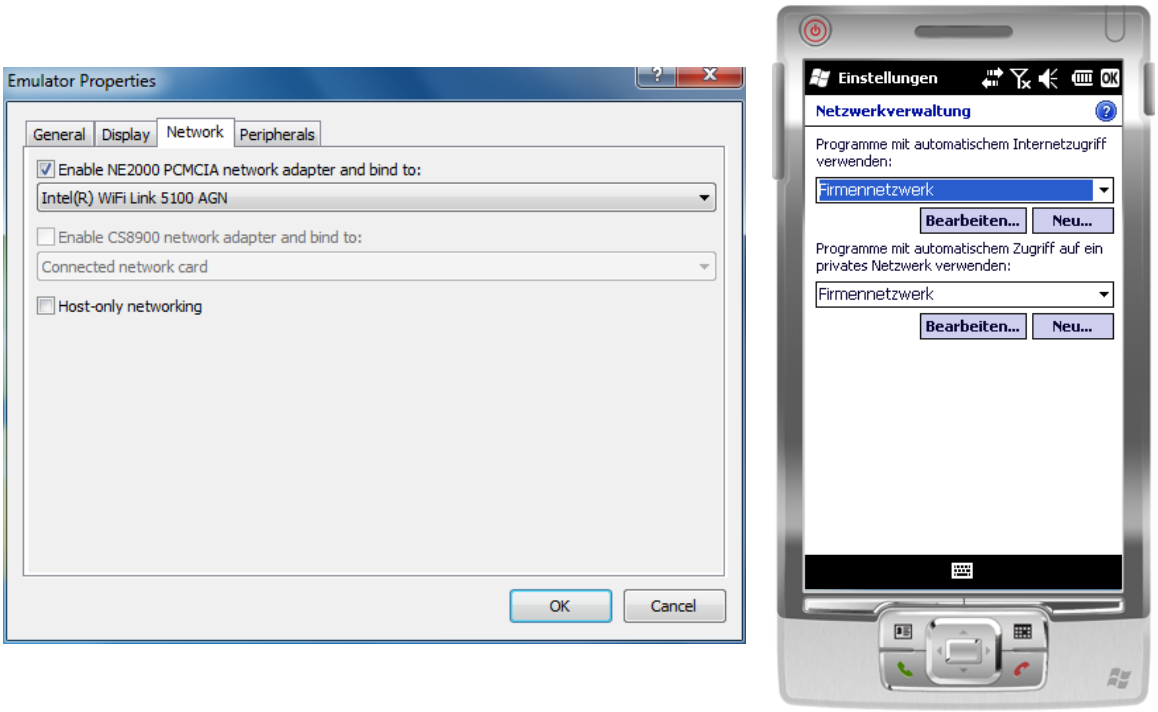
Problem
Windows Mobile und Windows Embedded Emulatoren empfangen keine UDP Multicast Datagramme. Der Multicast Empfang wird speziell beim Testen von WS-Discovery Features essenziell.

Lösung
Gemäss Release Notes vom Windows Embedded CE 6.0 R2 gibt es für die Emulatoren noch keine Lösung. Um Multicast Features zu testen ist man hier auf die Hardware angewiesen und muss die Applikation auf den richtigen Geräten ausführen.

14.2.2 Versand von TCP Paketen nicht möglich

Problem
Im Emulator können keine TCP Pakete versendet werden.

Lösung
Als erstes muss sichergestellt werden, dass für den Emulator ein Netzwerkadapter aktiviert wurde. Dies kann man im Emulator über die Menüpunkte „File → Configure… → Network“. Zusätzlich muss im Windows Mobile Betriebssystem das automatisch zu verwendende Netzwerk auf Firmennetzwerk gestellt werden. Dies kann man im Windows Mobile über „Start → Einstellungen → Verbindungen → Verbindungen → Erweitert → Netzwerke auswählen“.



14.3 Compact Framework 3.5

14.3.1 WinForms Buttons unterstützen kein ButtonDown/ButtonUp

Problem
Für die Umsetzung der Roboter Steuerung benötigt es einen Button, der einen ButtonDown bzw. ButtonUp Event unterstützt. Dies ist beim WinForms Button fürs Compact Framework leider nicht der Fall.

Lösung
Es wurde ein eigener `ImageButton` im RemoteController Projekt als User Control umgesetzt, der die beiden Events unterstützt.

14.3.2 Compact Framework lässt nur 2 offene TCP Connections zu

Problem
Bei der Entwicklung des Prototyps wurde festgestellt, dass der RemoteController nur zwei Web Service Aufrufe tätigen konnte und beim dritten Aufruf in ein Timeout gelangte.

Lösung
Nach einigen Recherchen wurde herausgefunden, dass das Compact Framework auf Windows Mobile 6.5 standardmässig nur zwei offene TCP Verbindungen zulässt. Das Problem lag aber daran, dass bei One-Way Calls der Requeststream nicht mehr richtig geschlossen wurde, worauf die Verbindung offen blieb. Schliesst man nach dem Absenden des Requests den Requeststream, so ist das Problem behoben.

14.4 Silverlight for Embedded

14.4.1 Error-Codes der XAML Runtime

Nicht jedes XAML, welches in Expression Blend kompiliert werden kann, übersteht die Initialisierung der XAML Runtime, welches lediglich einen Error Code wirft. Im Register Anhang ist eine Liste der Silverlight for Embedded Error Codes, welche wir aus dem Code extrahiert haben. Eine genauere Beschreibung der Konstanten ist unter *[URL2.10]* aufzufinden.

14.4.2 Aufspüren von Fehlern beim XAML parsing

Trotz Error Code weiss man besonders bei grösseren XAML Dateien nicht, wo der Fehler liegen könnte. Unsere Erfahrung war, dass Fehler oft beim Binden der Objekte respektive Event-Handlers auftraten. Dies geschieht in den beiden Methoden `BindEvents` und `BindObjects`, welche in den von XAML2CPP generierten Headerdateien T_xxx.h, wobei xxx für den Seitenname steht, zu finden sind. Beim durchsteppen der beiden Methoden mit dem Debugger sieht man genau, welches Element den Fehlercode auslöst.

Damit die DLL „debuggbar" wird, muss unter den Project Propertys → Configuration Properties → Debugging → Remote Executable auf die EXE-Datei konfiguriert werden, welche die DLL ladet, wie zum Beispiel
`%CSIDL_PROGRAM_FILES%\RC.RobotController\RC.RobotController.exe`

14.4.3 Schweizer Taschenmesser für Silverlight for Embedded: XAML2CPP

XAML2CPP generiert automatisch den Code, welcher benötigt wird, um die XAML Runtime zu initialisieren. Genauere Beschreibung zum Tool ist im Kapitel *9.3.4 Tools* zu finden.

14.4.4 Bugs in XAML2CPP 1.0.2.0

XAML2CPP erstellt für jedes Element, welches das Attribut `x:Name` definiert hat einen Pointer als Instanzvariable und bindet diesen in der Methode `BindObjects()` mit der Methode `findName(…)`. Innerhalb eines `ControlTemplate` darf kein `x:Name` gesetzt werden, da ansonsten das generierte `root->findName(…)` den Visual Tree durchläuft und das Element vom `ControlTemplate` nicht finden kann, da sich dieses Element im Visual Tree befindet und nicht im Logical Tree.

Workaround für XAML2CPP 1.0.2.0

`x:Name` kann durch `Name` (ohne x:) ersetzt werden. x:Name oder Name zu verwenden ist eher eine Codestyle Entscheidung als eine technische Entscheidung. Falls der Name nirgends referenziert wird, sollte dieses Attribut am besten entfernt werden.

Diesen Bug haben wir an den Entwickler von XAML2CPP weitergeleitet. In der Version 1.0.3.0 werden bei ControlTemplates keine C++ Bindings mehr generiert. Des Weiteren kann die Generierung einer Instanzvariable mit „_" als Prefix bei einem `x:Name` unterbunden werden.

14.4.5 Adressen der nativen Funktionspointer werden zur Laufzeit gelöscht

Problem

Bei der Entwicklung von Embedded Applikationen, bei denen die View mit Silverlight for Embedded und die Business Logik in C# geschrieben ist, müssen native Callbacks eingesetzt werden. Diese Callbacks werden in der View gebraucht um den C# Code über Eingaben des Benutzers zu informieren.

Ist das Zusammenspiel zwischen managed und native Code nicht richtig implementiert, so kann es vorkommen, dass zur Laufzeit die Adressen der nativen Funktionspointer auf die managed Delegates plötzlich gelöscht oder überschrieben werden. Die nativen Callbacks funktionieren ab diesem Zeitpunkt nicht mehr.

Lösung

Das managed Delegate, welches mittels P/Invoke dem nativen Code bekannt gemacht wird, muss im C# Code so initialisiert werden, dass dessen Adresse zur Laufzeit nicht geändert wird. Konkret bedeutet dies, dass alle managed Delegates die für die nativen Callbacks gebraucht werden als Instanzvariablen gespeichert werden müssen und nicht auf dem Stack sein dürfen.

14.5 Visual Studio Add-in Entwicklung

14.5.1 Auffinden von CommandBars

Ein Command kann an einer beliebigen Stelle im Visual Studio hinzugefügt werden, wie zum Beispiel im Kontextmenü des Solution Explorer. Dazu muss der Command bei der richtigen CommandBar hinzugefügt werden. Das Auffinden dieser CommandBar stellte sich als kein trivialer Prozess heraus. Abhilfe schafft folgender Trick:

In der Registry folgender Eintrag ändern:

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\9.0\General]
"EnableVSIPLogging"=dword:00000001
```

Drückt man nun ctrl+shift beim Klicken eines beliebigen Menüs erscheint folgende Visual Studio Debug Message:

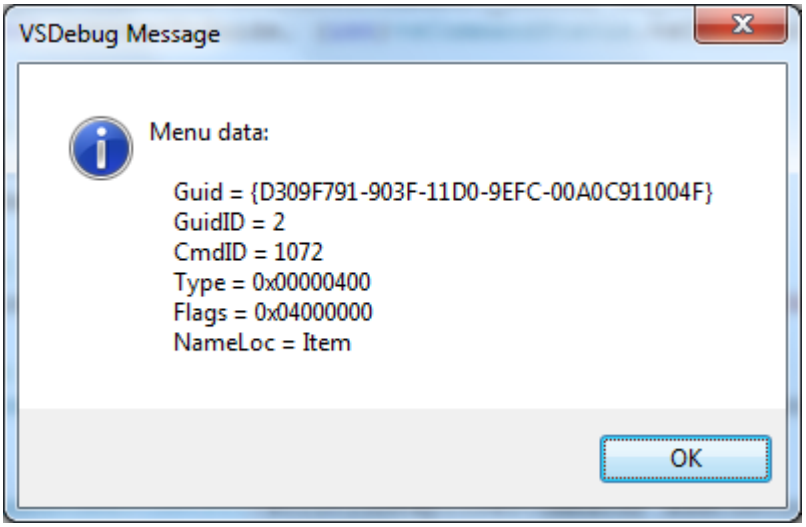


Abbildung 44: Visual Studio debug message

Mithilfe der Guid und der CmdID kann mit folgendem Code Snippet *[URL2.6]* die CommandBar eindeutig ermittelt werden:

```
private CommandBar FindCommandBar(Guid guidCmdGroup, uint menuId)
{
  // Make sure the CommandBars collection is properly initialized
  // before we attempt to use the IVsProfferCommands service.
  CommandBar menuBarCommandBar = ((CommandBars)_dte.CommandBars)["MenuBar"];

  // Retrieve IVsProfferComands via DTE's IOleServiceProvider interface
  IOleServiceProvider sp = (IOleServiceProvider)_dte;
  Guid guidSvc = typeof(IVsProfferCommands).GUID;
  Object objService;
  sp.QueryService(ref guidSvc, ref guidSvc, out objService);
  IVsProfferCommands vsProfferCmds = (IVsProfferCommands)objService;

  return vsProfferCmds.FindCommandBar(IntPtr.Zero, ref guidCmdGroup, menuId) as CommandBar;
}
```

Teil 6: Schlussfolgerungen

15 Bewertung der Ergebnisse

15.1 DPWS

Der DPWS-Stack ist eine wiederverwendbare Komponente in Form mehrerer DLL Dateien, welche in weiteren Projekten eingebunden werden können. Das Micro Framework besitzt einige gute Dokumentationen zum DPWS-Stack, welche dank der Portierung auch für unsere DPWS Version benutzt werden kann. Für das Generieren von Proxys und Stubs steht ein Kommandozeilen Tool zur Verfügung, welches ins Visual Studio integriert wurde. Dies ermöglicht anhand eines Interfaces oder einer WSDL Definition die zugehörigen Klassen, Interfaces und WSDL Definitionen bequem per Maus zu generieren.

15.2 Silverlight for Embedded

Das Silverlight for Embedded UI hat viel zu bieten: Schaltflächeneffekte, Animation bei Fensternavigation und diverse Transparenzeffekte. Damit diese Effekte flüssig ablaufen, werden mindestens 500Mhz und ein Grafikkartenbeschleuniger benötigt. Damit rechenintensivere Animationen flüssig dargestellt werden, muss auf dem OS Design Direct3D installiert sein.

Während der Implementation sind einige technische Erkenntnisse und Best Practices zusammengekommen, welche im Kapitel *14 Technische Erkenntnisse* dokumentiert wurden, die jedem Silverlight for Embedded Entwickler den Einstieg ungemein erleichtern werden.

15.2.1 Roboter Controller Showcase Anwendung

Von den 53 erfassten User Storys (Anforderungen) wurden 49 implementiert. Die beiden Applikationen Roboter Controller und Remote Controller benutzen beide den DPWS-Stack ausgiebig, so dass dieser zugleich erfolgreich auf seine Tauglichkeit getestet werden konnte. Die Anwendung eignet sich zur Präsentation der Kommunikationskomponente und dem Silverlight User Interface und wurde bereits vor der Abgabe der Arbeit bei Kunden zu Demozwecken verwendet.

16 Schlussfolgerungen

16.1 Silverlight for Embedded

Eine der grössten Herausforderung bei der Bachlorarbeit war die Implementation von Silverlight for Embedded. Bei Silverlight for Embedded handelt es sich im einen ersten Release von Microsoft, welcher zu Beginn der Arbeit erst 3 Monate jung war. Dies war beim Technologiestudium spürbar, da jegliche Suchresultate bei einem einzigen Blog von Valter Minute endeten.

Ein Teilziel der Bachelorarbeit war das Herausfinden, ob es möglich ist, eine Anwendung mit C# und dem Compact Framework 3.5 zu haben und dabei das UI mit Silverlight for Embedded zu entwickeln. Dabei sollen Erfahrungen gesammelt und dokumentiert werden.

16.1.1 Pro

Wie gewohnt von Silverlight kann das GUI komplett mit XAML deklarativ beschrieben werden. Die XAML Dateien lassen sich bequem mit Expression Blend 2 entwerfen. Im Vergleich zu MFC können Designer bei Silverlight mit Expression Blend arbeiten. So lassen sich auch viel ansprechbarere User Interfaces gestalten. Da ein C++ Projekt im Expression Blend 2 nicht geöffnet werden kann, ist im Kapitel *9.3.4 Tools für Silverlight for Embedded dokumentiert*, wie dies bewerkstelligt werden kann.

Gerade im Embedded Bereich ist Silverlight eine Revolution. Silverlight eröffnet viele Tore zum Entwickeln von ansprechbaren Userinterfaces.

16.1.2 Kontra

Die Implementation von Silverlight for Embedded ist in C++. Dies zieht einige Nachwirkungen mit sich. Unser Ziel war möglichst wenig C++ zu verwenden, so dass nur an wenigen Stellen auf den Komfort von .NET und C# verzichtet werden muss. Um die C++ Programmierung kommt man jedoch nicht herum. Silverlightspezifische Interaktion wie zum Beispiel das Starten eines Storyboards oder das Wechseln in einen anderen `VisualState` über den `VisualStateManager` muss in C++ erledigt werden. Events von Controls wie zum Beispiel `ButtonClick` müssen in C++ registriert, gefangen und mithilfe eines Funktionspointer an die managed Komponente weitergeleitet werden.

Eine Stärke vom eigentlichen Silverlight/WPF ist das DataBinding. Leider muss genau auf diese Funktionalität vollständig verzichtet werden. Die Kommunikation von managed zu native und wieder zurück zeigte sich als sehr schwerfällig und führt zu einer starken Kopplung zum C++ Code. Anstelle des DataBinding wurde ein Property Changed Mechanismus nachgebaut, welcher im Kapitel *9.3.3* Sequenzdiagramm Property Changed Mechanismus beschrieben ist. Bereits anhand des Sequenzdiagrammes ist die erhöhte Komplexität ersichtlich.

Des Weiteren muss für die XAML-Runtime extrem viel C++ Code geschrieben werden, damit das XAML korrekt geladen wird. Valter Minute, MVP für Windows Embedded Entwicklung, stellt auf seinem Blogg ein Programm XAML2CPP zur Verfügung, welches genau diese Arbeit übernimmt. Microsoft verspricht eine Verbesserung bezüglich Codegeneration für den nächsten Release.

16.1.3 Schlussfolgerung

Selbst ein fortgeschrittener C# Entwickler mit guten C++ Kenntnissen wird sich auf ein erstes Erfolgserlebnis mit Silverlight for Embedded gedulden müssen. Die knappe MSDN Dokumentation reichte uns nicht aus, worauf ein weiterer Anlauf mit XAML2CPP gestartet wurde. Im Kapitel *14.4 Silverlight for Embedded* sind einige Stolpersteine Dokumentiert.

Wichtige Komponenten wie zum Beispiel DataBinding müssen nachgebaut werden. Gerade der Durchstoss von native zu managed und umgekehrt birgt einige Stolpersteine, welche viel Zeit kosten. Sobald dieser Durchstoss erreicht ist, kann relativ effizient gearbeitet werden. Unsere Showcase Anwendung besitzt bereits einen möglichst „generischen“ Ansatz, welcher sich auch auf andere Projekte adaptieren lässt.

Aus der MFC Entwickler Sicht wird Silverlight for Embedded ein riesen Sprung sein, als .NET Silverlight Entwickler benötigt man doch eine beachtliche Einarbeitungszeit, weil bis auf das Gestalten des UI ist der Wiedererkennungswert zum üblichen Silverlight gering ausfällt.

Teil 7: Anhang

17 Verzeichnisse

17.1 Abbildungsverzeichnis

17.2 Tabellenverzeichnis

17.3 Literaturverzeichnis

[Gamma95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. 1995 *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc.

[Kühner08] Kühner, J 2008 Expert .NET Micro Framework, Apress
http://www.apress.com/book/view/159059973x

[OASIS] OASIS DPWS 1.1 Spezifikation
http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[Url2.1] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html

[Url2.2] WSDAPI
http://msdn.microsoft.com/en-us/library/aa826001(VS.85).aspx

[Url2.3] DPWS im :NET Micro Framework
http://msdn.microsoft.com/en-us/library/ee435393.aspx

[Url2.4] WS4D.org Java Multi Edition DPWS-Stack
http://sourceforge.net/projects/ws4d-javame/

[Url2.5] WS4D-gSOAP
http://trac.e-technik.uni-rostock.de/projects/ws4d-gsoap

[Url2.6] Ermittlung von CommandBars für Visual Studio Add-in Entwicklung
http://blogs.msdn.com/b/dr._ex/archive/2007/04/17/using-ivsproffercommands-to-retrieve-a-visual-studio-commandbar.aspx

[Url2.7] .NET Micro Framework Porting Kit
http://www.microsoft.com/downloads/details.aspx?FamilyID=16fa5d31-a583-4c0d-af74-f4d5e235d5bc

[Url2.8] XAML2CPP Tool von Valter Minute
http://geekswithblogs.net/WindowsEmbeddedCookbook/category/10948.aspx

[Url2.9] MVVM Pattern
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[Url2.10] Silverlight for Embedded Error Codes
http://msdn.microsoft.com/en-us/library/ee501497.aspx

Alle Weblinks sind im Juni 2010 auf Ihre Erreichbarkeit überprüft worden.

Aufgabenstellung

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

Roboter Controller

Aufgabenstellung für Stefan Züger und Tobias Zürcher

Aufgabenstellung gemäss beiliegendem Dokument von Zühlke.
Die definitive Aufgabentstellung wird nach dem Vorliegen des Projektauftrages festgelegt.

Resultate

Software und Dokumentation gemäss HSR-Richtlinien.

Projektpartner

Auftraggeber

Zühlke Engineering AG
Wiesenstrasse 10a
8952 Schlieren (Zürich)
Dipl. Ing. Mario Klessascheck (Auftraggeber)
E-Mail: mkl@zuehlke.com
Dipl. Ing. Michael Koster (Ansprechpartner für technische Fragen)
E-Mail:: mko@zuehlke.com

Studierende

Stefan Züger
E-Mail: szueger@hsr.ch
Tobias Zürcher
E-Mail: tzuerche@hsr.ch

Betreuung HSR

Hansjörg Huser
E-Mail: hhuser@hsr.ch
Tel 079 276 43 09

Jürg Jucker
E-Mail: jjucker@hsr.ch

Projektabwicklung

Termine:

- Beginn der Arbeit: M**o., 22. Feb. 2010**
- Abgabetermin Kurzfassung/Poster/Mgmt-Summary zum Review: 11.06.2010
- Abgabetermin (ink. Poster): 18.06.10, 12.00 Uhr
- **HSR-Forum, Vorträge und Präsentation der Bachelor- und Diplomarbeiten, 25.6.2010 13 bis 18 Uhr**
- Mündliche BA-Prüfung : genaues Datum folgt (21.06. - 31.08.2010)
- Zwischenbesprechung/Review mit Auftraggeber nach Projektplan

Arbeitsaufwand

Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet. Dies entspricht einer Arbeitsleistung von mind. 360 Stunden pro Student.

Hinweise für die Gliederung und Abwicklung des Projektes:

Gliedern Sie Ihre Arbeit in 4 bis 5 Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

Folgende Teilschritte bzw. Meilensteine sollten Sie in Ihrer Planung vorsehen:

Schritt 1: Projektauftrag inkl. Projektplan (mit Meilensteinen),

- Meilenstein 1: Review des Projektauftrages abgeschlossen. Projektauftrag von Auftraggeber und Dozent genehmigt
  Letzter Meilenstein: Systemtest abgeschlossen
  Termin: ca. eine Woche vor Abgabe
- Entwickeln Sie Ihre SW in einem iterativen, inkrementellen Prozess: Planen Sie möglichst früh einen ersten lauffähigen Prototypen mit den wichtigsten und kritischsten Kernfunktionen. In die folgenden Phasen können Sie dieses Kernsystem schrittweise ausbauen und testen.
- Falls Sie in Ihrer Arbeit neue oder Ihnen unbekannte Technologien einsetzen, sollten Sie parallel zum Erarbeiten des Projektauftrages mit dem Technologiestudium beginnen.
- Setzen Sie konsequent Unit-Tests ein! Verwalten Sie ihre Software und Dokumente auf einem SVN-Repository. Stellen Sie sicher, dass der/die Betreuer jederzeit Zugriff auf das Repository haben und dass das Projekt anhand des Repositories jederzeit wiederhergestellt werden kann.
- Achten Sie auf die Einhaltung guter Programmier- und Designprinzipien
  - DRY, high cohesion, loose coupling, etc.
  - Clean Code!
- Halten Sie sich im Übrigen an die Vorgaben aus dem Modul SE-Projekt.

Projektadministration

- Führen Sie ein individuelles Projekttagebuch aus dem ersichtlich wird, welche Arbeiten Sie durchgeführt haben (inkl. Zeitaufwand). Diese Angaben sollten u.a. eine individuelle Beurteilung ermöglichen.
- Dokumentieren Sie Ihre Arbeiten laufend. Legen Sie Ihre Projektdokumentation mit der aktuellen Planung und den Beschreibungen der Arbeitsresultate elektronisch in einem Projektordner ab. Dieser Projektordner sollte jederzeit einsehbar sein (z.B svn-Server oder File-Share).

Inhalt der Dokumentation

Bei der Abgabe muss jede Arbeit folgende Inhalte haben:

- Dokumente gemäss Vorgabe: https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html
- Aufgabenstellung
- Technischer Bericht
- Projektdokumentation
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.

Form der Dokumentation:

- Bericht (Struktur gemäss Beschreibung) in Ordner(1 Exemplar für HSR)
- Alle Dokumente und Quellen der erstellten SW auf CD, CD's sauber angeschrieben (3 Ex.).

Fortschrittsbesprechung:

Regelmässig findet zu einem fixen Zeitpunkt eine Fortschrittsbesprechung statt.
Teilnehmer: Dozent und Studenten, bei Bedarf auch Vertreter der Auftraggeber
Termin: jeweils xxx, Raum 6.010 (Abweichungen werden rechtzeitig kommuniziert)
Traktanden

- Was wurde erreicht, was ist geplant, welche Probleme stehen an
- Review von Code/Dokumentation (Abgabe jeweils einen Tag vor dem Meeting

Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden.
Sie erstellen zu jeder Besprechung ein Kurzprotokoll, welches Sie spätestens 2 Tage nach der Sitzung per e-mail an den Betreuer senden.

Rapperswil, 22. Feb. 2010
Hansjörg Huser

Thema Diplomarbeit HSR - Zühlke Roboter Controller

Mario Klessascheck
Februar 2010

Die HSR Hochschule für Technik Rapperswil hat Zühlke für ein Diplomthema im Bereich Softwareentwicklung angefragt.

Als mögliches Thema schlägt Zühlke die Teilentwicklung einer Schulungsplattform für embedded Projekte vor. Die Teilentwicklung beinhaltet die Entwicklung eines managed Webserver und dem HMI für einen Roboter Controller (WIN CE) sowie einem GUI für einen Romote Controller (WIN Mobile).

Dok. 1
Version 1
Datum 05.02.2010

Zühlke Engineering AG
Wiesenstrasse 10a
8952 Schlieren (Zürich)
Schweiz

Telefon +41 44 733 6611
Telefax +41 44 733 6612

info@zuehlke.com
www.zuehlke.com

Verteiler

Firma	Name
HSR	Prof. Hansjörg Huser, Jürg Jucker
Zühlke	Michael Koster, Daniel Tobler, Mario Klessascheck
Studierende	Tobias Zürcher, Stefan Züger

Inhaltsverzeichnis

1. Einleitung

Im Rahmen einer Diplomarbeit an der HSR sollen Teile einer Schulungsplattform entwickelt werden. Die Schulungsplattform entwickelt Zühlke für interne Schulungen. Aus dieser Schulungsplattform wurden abgeschlossene Arbeitspakete extrahiert, die im Rahmen dieser Diplomarbeit zu entwickeln sind.

2. Aufgabenstellung

Für den Aufbau einer Schulungsplattform soll eine Steuerung entwickelt werden, mit der ein Spielzeugroboter gesteuert werden kann. Die Plattform wird für interne Schulungszwecke im embedded Bereich verwendet. Weiterhin dient die Plattform zum Testen neuer GUI Technologien und Kommunikationskonzepten. Zum Aufbau dieser Schulungsplattform müssen bestimmte Grundarbeiten durchgeführt werden. Diese Arbeiten werden folgend beschrieben.

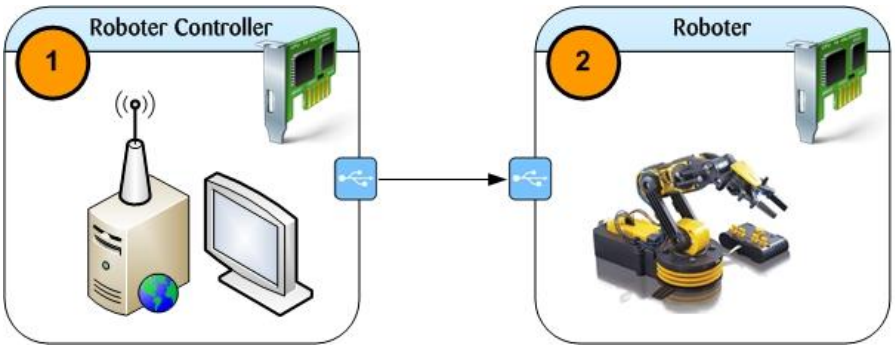


Abbildung 1 Systemarchitektur Roboter und Roboter Controller

(1) Roboter Controller

Der Roboter Controller besteht aus einer 32bit embedded Plattform mit einem ARM Controller. Als Betriebssystem wird Windows Embedded CE 6.0 R3 verwendet. Für die Bedienung des Roboters wird ein Touchscreen angeschlossen. Die Kommunikation zwischen Roboter Controller (1) und der Steuerplatine (2) des Roboters erfolgt über eine USB Schnittstelle. Für die Kommunikation nach aussen enthält die embedded Plattform ein LAN und ein WLAN Interface.

Aufgabe

Für den Roboter Controller soll ein managed Webserver entwickelt und implementiert werden. Der Webserver stellt Webservices für die Remotesteuerung und die Betriebsdatenerfassung zur Verfügung. Die Betriebsdatenerfassung bietet dem Service eine Monitoringschnittstelle für Maintenance Aufgaben. Über den Webservice kann für bestimmende Betriebsparamter ein Monitoring mit Überwachung aktiviert werden. Die lokale Steuerung erfolgt über ein Touchscreen. Das GUI auf den Touchscreen soll modernen HMI Konzepten entsprechen. Als Technologie für das GUI soll nach Möglichkeit Silverlight für embedded Systeme verwendet werden.

(2) Roboter

Der Roboter besitzt eine USB Schnittstelle für die Steuerung. Dafür stehen für den PC bereits Treiber zur Verfügung. Für Windows Embbedded CE liegen keine Treiber vor. Die Treiber sind nicht Bestandteil der Arbeit und werden von Zühlke geliefert. Der Roboter wird von Zühlke zur Verfügung gestellt. Für die Entwicklung des Webservers und der Steuerung muss der Roboter noch nicht zur Verfügung stehen.

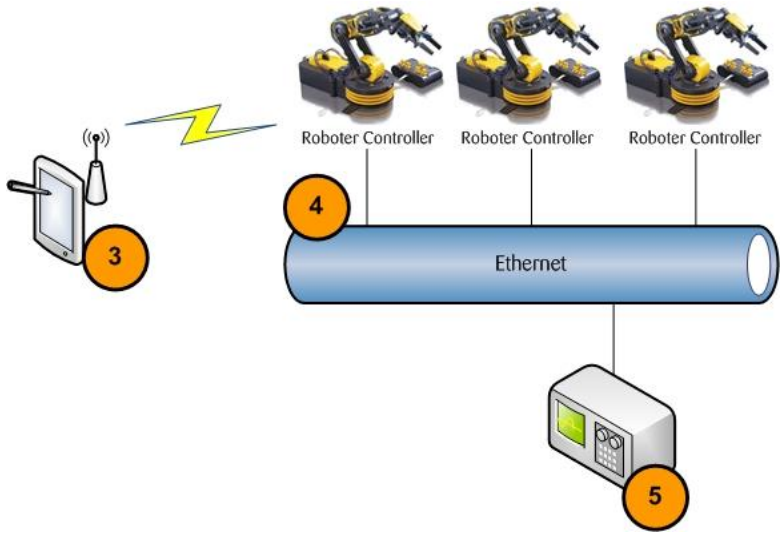


Abbildung 2 Systemkonzept Remote Control

(3) Remote Controller

Der Remote Controller ist ein Handy mit dem Betriebsystem **Windows Mobile 6.5**. Über den Remote Controller (3) kann ein Roboter (4) ferngesteuert werden. Wenn mehrere Roboter vorhanden sind, muss ein Roboter ausgewählt werden. Die Steuerung erfolgt über den Webservice des Roboter Controllers (1). Für Servicezwecke sollen auf dem Remote Controller die Betriebsdaten der verschiedenen Roboter angezeigt werden können. Wenn ein Roboter einen kritischen Fehlerzustand besitzt, wird der Fehler über den Webserver des Roboter Controllers (1) gemeldet. Der Remote Controller muss in der Lage sein, den Alarm zu empfangen und entsprechend darzustellen.

Aufgabe

- GUI für Fernbedienung Compact Framework
- Anzeige Betriebsdaten
- Alarmanzeige

(4) Roboter

Mehrere Roboter können über das LAN Interface zu einem Roboternetz hinzugefügt werden. Über das Roboternetz kann ein Roboter im Fehlerfall den „Notaus“ für alle Roboter auslösen. Die Monitoringeinheit (5) ist ein PC, über den die Betriebsdaten aller Roboter dargestellt werden.

(5) Monitoring Zentrale

Die Zentrale dient zur Betriebsdatenerfassung aller Roboter. Die Betriebsdaten werden gespeichert und erlauben eine Überwachung. Die Monitoringzentrale ist ein PC. Die Zentrale ist kein Bestandteil dieser Arbeit und wir nur der Vollständigkeit wegen dargestellt.

3. Nichtfunktionale Aufgaben

Neben der eigentlichen Entwicklung der Teilkomponenten aus Kapitel 2 müssen folgende Aufgabe erledigt werden:

- Requirementsengineering: Dokumentation mit Anforderungen erstellen
- Entwicklungsdokumentation: Architektur, Design, Testprotokolle
- Tutorial als Einstieg für die Applikationsentwicklung unter Windows Mobile 6.5

4. Ergebnisse

Die Ergebnisse der Arbeit (Dokumente, Code) stehen Zühlke und der HSR für die weitere Nutzung uneingeschränkt zur Verfügung.

5. Umfang

Start der Arbeit: 22.2.2010

Ende der Arbeit: 18.6.2010

Weitere Informationen zu den Terminen finden sich in der HSR-Rahmenaufgabenstellung.

6. Betreuung

Die Betreuung HSR:

- Prof. Hansjörg Huser: Institutsleiter INS, Dozent für Informatik
- Jürg Jucker. Dipl Ing, Leiter .NET Kompetenzzentrum

Die Betreuung Zühlke:

- Dipl. Ing. Michael Koster: Windows Embedded/Mobile Experte, MVP (technische Konsultation)
- Dipl. Ing. Mario Klessascheck: Business Unit Leiter (Vereinbarung)

Verantwortlichkeit

Da die Zühlke Mitarbeiter in anderen Projekten tätig sind, ist der Zeitaufwand für die Betreuung so gering wie möglich zu halten. Zühlke steht für technische Konsultationen, Reviews und für die Anforderungsphase zur Verfügung.

Ergänzende Spezifikationen

Project:	Diplomarbeit Roboter Controller
Customer:	HSR / Zühlke
Project Nr:	-
Document Nr:	-
Version:	1.0
Date:	26.02.2010

Inhalt

1. Einleitung

1.1 Ziel

Die folgenden Anforderungen, Schnittstellen und deren Spezifikationen beziehen sich auf die Subsysteme Webserver und GUI (Roboter Controller) und Webservice und GUI (Remote Controller). Das Dokument soll als Grundlage für die Erstellung der Software Requirement Spezifikationen (SRS) verwendet werden.

1.2 Projekt Scope

Es werden im Wesentlichen zwei Softwareapplikationen (Roboter Controller und Remote Controller) entwickelt. Der Roboter Controller ist die zentrale Steuereinheit für einen Roboter und besteht aus den Subsystemen Treiber (Axen Steuerung), GUI (Bedienung) und Webservice (Kommunikation). Die Applikation läuft auf einem WIN CE Rechner. Der Romote Controller dient als Fernsteuerung und Serviceplattform für den Roboter. Der Remote Controller besteht aus den Subsystemen GUI für die Fernsteuerung und Webservices für die Kommunikation mit dem Roboter Controller. Die Softwareentwicklung bezieht sich ausschliesslich auf die Software Applikationen für den Roboter Controller und den Remote Controller.

1.3 Referenzen

[1]Grobe Beschreibung der Funktion des Roboter Controllers innerhalb der Schulungsplattform
Beschreibung Thema Roboter Controller HSR.doc

[2]WS-I Basic Profile:
http://www.ws-i.org/Profiles/BasicProfile-1.1.html

[3]SOAP 1.2:
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[4]HTTP 1.1:
http://www.ietf.org/rfc/rfc2616.txt

[5]WSDL 1.1:
http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[6]MSBuild:
http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx

2. Beschreibung

Zurzeit existiert keine managed Implementierung eines DPWS-Stacks (Device Profile for Webservices) für das Compact Framework. Innerhalb dieser Arbeit soll unter Anderem ein DPWS-Stack entwickelt werden, den Zühlke in Zukunft in für die Implementierung von Webservices in anderen Projekten verwenden kann.

Die Implementierung des DPWS-Stacks wird in zwei Beispielapplikationen (PoC) getestet. Die Applikationen werden in einer Schulungsplattform weiter verwendet und sind Bestandteil dieser Arbeit. Die Schulungsplattform verwendet Zühlke für die Aus- und Weiterbildung zu technologiespezifischen Themen betreffend Windows CE und Windows Mobile.

2.1 Benutzer

Benutzer des DPWS-Stacks sind ausschliesslich Software Entwickler, die im embedded und mobile Umfeld tätig sind. Teile der Gesamtapplikation sollen auch als Showcase für die Demonstration bei einem Kunden verwendet werden.

3. System Features

3.1 Funktionale Anforderungen

3.1.1 Roboter Controller

Der Roboter Controller besteht aus einer 32bit embedded Plattform mit einem ARM Controller. Als Betriebssystem wird Windows Embedded CE 6.0 R3 verwendet. Die Bedienung des Roboters erfolgt über Touchscreen und Tastatur. Die Kommunikation zwischen Roboter Controller und der Steuerplatine des Roboters erfolgt über eine USB Schnittstelle. Für die Kommunikation nach aussen enthält die embedded Plattform ein LAN und ein WLAN Interface.

Aufgabe

Auf dem Roboter Controller stellt ein Webserver Services für die Remotesteuerung und die Betriebsdatenerfassung zur Verfügung. Die Betriebsdatenerfassung bietet einem Service-Benutzer eine Monitoringschnittstelle für Maintenance Aufgaben. Für die einzelnen Betriebsparameter kann der Benutzer auswählen, ob die Überwachung aktiviert werden soll. Für die Überwachung muss der Benutzer Alarmgrenzen hinterlegen können. Falls eine der definierten Alarmgrenzen überschritten wird, sendet der Roboter Controller einen Alarm zum Remote Controller. Über den Roboter Controller soll es möglich sein, wiederholbare komplexe Bewegungsabläufe zu programmieren (Teaching). Die Bwegungsabläufe werden gespeichert und können später wieder aufgerufen werden. Die Bewegungsabläufe müssen auch über den Remote Controller aufgerufen und gestartet werden können.

- GUI für Bedienung
- Aufzeichnen von Bewegungsabläufen (Teaching)
- Aufruf und Start von Bewegungsabläufen
- Anzeige Betriebsdaten
- Alarmlog und Anzeige
- Der Roboter Controller soll auch als Remote Controller verwendet werden.

3.1.2 GUI Remote Controller

Der Remote Controller ist ein Handy mit dem der Roboter ferngesteuert wird. Wenn mehrere Roboter vorhanden sind, muss ein Roboter ausgewählt werden. Die Steuerung erfolgt über den Webservice des Roboter Controllers. Es kann immer nur eine Roboter gesteuert werden. Für Servicezwecke sollen auf dem Remote Controller die Betriebsdaten der verschiedenen Roboter angezeigt werden können. Wenn ein Roboter einen kritischen Fehlerzustand besitzt, wird der Fehler über den Webserver des Roboter Controllers gemeldet. Der Remote Controller muss in der Lage sein, den Alarm zu empfangen und entsprechend darzustellen. Über den Remote Controller müssen gespeicherte Bewegungsabläufe abgerufen und ausgeführt werden können.

Aufgabe

- GUI für Fernbedienung
- Anzeige von Betriebsdaten
- Alarmlog und Anzeige
- Aufruf und Start von Bewegungsabläufen

4. Interfaces

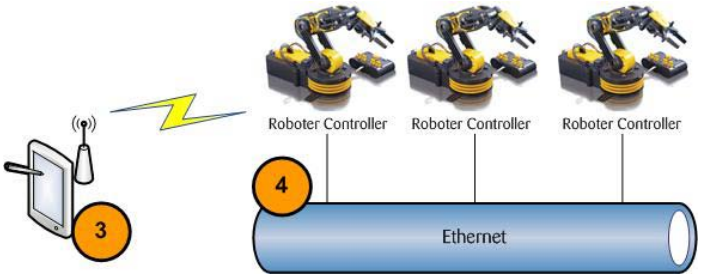


Abbildung 1 Interfaces Roboter und Remote Controller

4.1 Benutzer Schnittstellen

Remote Controller (3) und Roboter Controller (4) haben ein Touchinterface für die Fernsteuerung des Roboters. Auf beiden Controllern (3, 4) können die Betriebsdaten eines Roboters dargestellt werden. Auf dem Roboter Controller (4) kann der Roboter zusätzlich über eine angeschlossene Tastatur bedient werden. Das Aufzeichnen und Speichern von Bewegungsabläufen erfolgt auf dem Roboter Controller (4).

4.2 Hardware Schnittstellen

Die Ansteuerung der Axen des Roboters erfolgt über eine USB Schnittstelle mit einer Steuerplatine. Die Steuerplatine muss nicht entwickelt werden. Der OS-Treiber für die USB-Steuerplatine wird von Zühlke entwickelt.

4.3 Software Schnittstellen

Die Kommunikation zwischen Remote Controller (3) und Roboter Controller (4) erfolgt via Webservices über HTTP. Der Remote Controller (3) sendet die Steuerbefehle via Webservice zum Roboter Controller. Der Roboter Controller empfängt die Steuerkommandos und sendet die Kommandos über die OS-Treiber zur USB-Steuerplatine. Der Benutzer kann die Betriebsdaten eines Roboters auf dem Remote Controller via Webservice vom Roboter Controller abrufen.

Anforderungen DPWS-Stack:

- Wiederverwendbare Komponente , die den DPWS-Stack (Client und Server) implementiert
- Integration in Visual Studio analog ASP.NET/WCF
- Integration in MSBUILD/TeamBuild

4.4 Protokolle und Schnittstellen

Für das USB Interface werden Treiber entwickelt, die nicht im Scope dieser Aufgabe liegen. Das Interface sowie das Protokoll für die USB-Treiber aus Sicht der Applikation sind noch zu spezifizieren und sind Teil dieser Arbeit. Für die Remote-Kommunikation werden Webservices verwendet.

5. Nicht funktionale Anforderungen

- Um die Optimale Lösung hinsichtlich der nichtfunktionalen Anforderungen wie Performance oder Wiederverwendbarkeit zu erfüllen, sollen verschiedene Lösungsvarianten diskutiert werden.
- GUI auf dem Roboter Controller mit Silverlight for embedded
- SOAP Spec. 1.2

- Managed Implementierung des Interfaces für den DPWS-Stack.
- Architektur M-V-V-M Pattern für die Wiederverwendbarkeit der Businesslogik
- UML-Konforme Projektdokumentation
- Das GUI auf den Touchscreen muss modernen HMI Konzepten (Useability) entsprechen.

5.1 Performance

Da die Applikation Roboter Controller auf einem embedded System läuft, ist es wichtig, dass die Implementierung hinsichtlich Speicher und CPU Belastung optimiert wird.

5.2 Qualität

- Entwurfsrichtlinien einhalten (siehe FxCop Developer Studio)
- Dokumentation der Interfaces
- Keine Compilerfehler
- Keine Abstürze

5.3 Andere Anforderungen

- Dokumentation für den Benutzer
- Kleines Tutorial als Einstieg für die Applikationsentwicklung unter Windows Mobile 6.5

6. Revision History

Version	Date	Author(s)	Description
1.0	22.02.2010	Mario Klessascheck	Erste Version

Requirements Analyse

Version 1.2

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

Revision

Version	Status	Datum	Beschreibung/Änderung	Autor
0.1	Draft	04.03.2010	Erste Version Fragenkatalog	TZ, SZ
0.2	Draft	09.03.2010	Antworten zum Fragenkatalog hinzugefügt	SZ
0.3	Draft	10.03.2010	Grundstruktur, Projektübersicht & Scope	TZ
0.4	Draft	10.03.2010	Anforderungen Roboter Controller	TZ
0.5	Draft	10.03.2010	Anforderungen Remote Controller begonnen	TZ
0.6	Draft	11.03.2010	Struktur der Anforderungen umgebaut, Anforderungen DPWS geschrieben, Anforderungen Roboter Controller erweitert	SZ
0.7	Draft	14.03.2010	Anforderungen Remote Controller erweitert	SZ
0.8	Draft	15.03.2010	Einige kleinere Korrekturen, Schnittstellen vom Remote Controller	TZ
0.9	Draft	15.03.2010	Priorisierung aller Requirements, Requirements für Integration in Visual Studio	SZ
0.10	Draft	15.03.2010	Use-Case Diagramme eingefügt, Review Priorisierungen	TZ
1.0	Released	15.03.2010	Dokument für Review fertiggestellt	SZ
1.1	Released	23.03.2010	Änderungen gemäss Feedback von MKL	SZ/TZ
1.2	Released	12.04.2010	Änderungen gemäss Feedback von MKL. Schnittstellenanforderungen von Roboter Controller als Tabelle	SZ

Teil 1: Interview

1 Roboter Controller

1.1 Webservice

In der Aufgabenstellung wird oft von einem „Webserver“ gesprochen. So wie wir DPWS verstanden haben, wird kein zusätzlicher Webserver benötigt. Korrekt? [1]

> Richtig! Es braucht nur noch die Implementierung des DWPS Stacks und kein Webserver mehr.

Welche Komponenten des DPWS-Stack sind umzusetzen? [2]

- Messaging (URI, UDP, HTTP, SOAP, WS-Addressing, Attachements)
- Discovery
- Description (Characteristics, Hosting, WSDL, WS-Policy)
- Eventing
- Security

> Zühlke braucht alle Komponenten des DPWS-Stacks.

.NET Micro Framework besitzt bereits eine managed Implementation des DPWS-Stacks. Ist eine eigene Neu-implementation in C# vorgesehen? Portierung? Wrapper für native Implementation? [3]

> Source von Micro Framework könnte als Vorlage dienen um eine Portierung aufs Compact Framework durchzuführen. Evtl. muss zuerst aber eine Variantenanalyse getätigt werden, was alles möglich ist.

1.2 UI auf Embedded System

Roboter Controller kann über Touchinterface und Tastatur gesteuert werden. Wird die Tastatur lediglich für Texteingaben verwendet? Gibt es exklusive Tastaturfunktionen? [4]

> Tastatursteuerung des Roboters soll auch möglich sein. Tastaturbelegung ist egal.

1.3 Roboter Steuerung

Welche ansteuerbaren Befehle kann der Roboter ausführen? [5]

Roboter versteht grundsätzlich zwei unterschiedliche Befehle: Zum einen der Befehl um eine der fünf vorhandenen Achsen zu steuern. Die Schnittstelle dazu lautet ungefähr „moveAxis(motor, direction)“. Man sagt dem Roboter welcher der 5 Motoren (M1 – M5) bewegt werden soll und in welche Richtung er drehen soll (left/right). Der zweite Befehl ist das Ein-/Ausschalten eines LED Lämpchens (LED1), welches vorne am Greifarm montiert ist.

Es ist abzuklären, wie die anhaltende Bewegung des Roboters umgesetzt wird. Dazu gibt es zwei Varianten. Entweder bewegt er sich inkrementell beim Knopfdruck um einen bestimmten Wert oder er bewegt sich konstant, solange der Knopf betätigt wird.

Die Steuerung über das Touch Interface soll direkt den Roboter ansteuern, ohne über den Webservice zu gehen.

Es ist abzuklären, ob der Webservice Aufruf synchron oder asynchron implementiert werden soll.

Welche Fehlerzustände gibt es? [6]

Pro Achse/Motor gibt es einen generellen Fehlerzustand. Dieser enthält eine einfache Fehlernummer. Der Fehlerzustand pro Achse wird in 1 Byte abgespeichert.

Zusätzlich kann der Benutzer des Touch Interfaces ein Timeout konfigurieren, wie lange eine Taste ununterbrochen gedrückt werden darf. Wird diese Zeit überschritten, so wird auch ein Fehler ausgelöst, der ein Notaus für alle Roboter im Robotern etz zur Folge hat.

Es soll ein Log über alle Fehler geführt werden.

Welche Alarmgrenzen kann der Benutzer bestimmen? [7]

Der Benutzer des Touch Interfaces kann pro Achse/Motor eine Alarmgrenze setzen, die sagt, wie lange ein Motor benutzt werden kann, bis er einen Service benötigt. Wird eine solche Grenze überschritten, so erhält der Benutzer einen Alarm mit einem entsprechenden Service Eintrag.

Welche Betriebsdaten besitzt ein Roboter? Hat der Roboter Sensoren, die ausgelesen werden können? (gemäss Sitzung: nein) [8]

Der Roboter besitzt selber keine Daten, die ausgelesen werden könnten. Der Roboter Controller führt aber die aufgelaufene Betriebszeit aller Motoren.

Annahme: Ein Roboter Controller steuert genau einen Roboter? [9]

Korrekt! Spezialfall, dass ein Roboter Controller auch als Remote Controller agieren kann. Dann ist er selber nicht direkt an einem Roboter angehängt, sondern steuert einen entfernten Roboter via DPWS.

In einem Roboternetz kann ein einzelner Roboter den Notaus für alle Roboter auslösen. Wer kann den Normalzustand wiederherstellen? Wo wird der Notaus aufgehoben? [10]

Auf dem Touch Interface wird ein Notaus Taster dargestellt, mit dem man ein Notaus fürs Roboternetz auslösen kann. Der Notaus kann via GUI aufgelöst werden oder er wird nach einem bestimmten Timeout automatisch aufgehoben. Notaus soll auch über Remote Controller möglich sein.

2 Remote Controller

Können Bewegungsabläufe nur abgerufen und gestartet oder auch neu aufgezeichnet werden? (Auf dem Remote Controller) [11]

Auf Remote Controller können keine Bewegungsabläufe aufgezeichnet werden.

Muss der Zugriff auf den Roboter Controller geschützt sein oder genügt WLAN Verschlüsselung? [12]

> Es soll ein Schutz nach Geräteklassen vorgenommen werden, damit bestimmt nur ein Remote Controller zum Roboter Controller verbinden kann. Roboter Controller sollen auch nur auf Discovery Anfragen von Remote Controllern antworten.

Ein Roboter kann zur gleichen Zeit nur von jemandem gesteuert werden (Ausnahme: Notaus). Soll diese „Blockierung“ aufhebbar sein? (im Falle eines Absturzes des Remote Controller) [13]

> Timeout ist nicht notwendig. Manuelle Aufhebung über Touch genügt. Roboter Controller und Remote Controller sollen sich immer die Adressen des letzten gültigen Partners speichern. Wenn dann ein unerwarteter Verbindungsabbruch geschieht und die beiden sich irgendwann wieder sehen, soll ein automatischer Verbindungsaufbau geschehen.

3 Nicht funktionale Anforderungen

FxCop Developer Studio/Style Cop: Konfiguration bzw. Richtlinien von Zühlke? [14]

> Zühlke hat keine eigenen Richtlinien oder Konfigurationen von FxCop. Der Standard soll hier verwendet werden.

MVVM gefordert, Silverlight for Embedded unterstützt jedoch kein Databinding (MVVM ohne Databinding problematisch)? [15]

> Aus den oben genannten Gründen ist MVVM keine Anforderung mehr.

4 Weitere Fragen

Verständnisfrage zum Roboternetz: Müssen Roboternetze innerhalb eines LAN Segment konfigurierbar sein, oder 1 LAN Segmet = 1 Roboternetz? [16]

> 1 LAN Segment = 1 Roboternetz. Keine Konfiguration notwendig.

Proxy/Stub Generierung, Visual Studio Integration und MSBuild/TeamBuild Integration sind je nach Funktionsvielfalt aufwändig. Was sind die Minimalanforderungen? [17]

> Proxy/Stub Generierung ist ganz klar eine Anforderung, so wie es mit dem SvcUtil von WCF möglich ist. Anforderungen bezüglich MSBuild/TeamBuild muss Mario mit Michael abklären.

Teil 2: Software Requirements Specification

5 Einleitung

5.1 Zweck

Das Dokument spezifiziert die Anforderungen zur Bachelorarbeit Roboter Controller, welche in Zusammenarbeit mit der Firma Zühlke Engineering AG realisiert wird.

5.2 Projekt Scope

Im Wesentlichen besteht das Projekt aus zwei Softwarekomponenten: dem Roboter Controller (1) und dem Remote Controller (3). Der Roboter wird über eine Treiberschnittstelle (2) gesteuert, welcher von Zühlke geliefert wird.

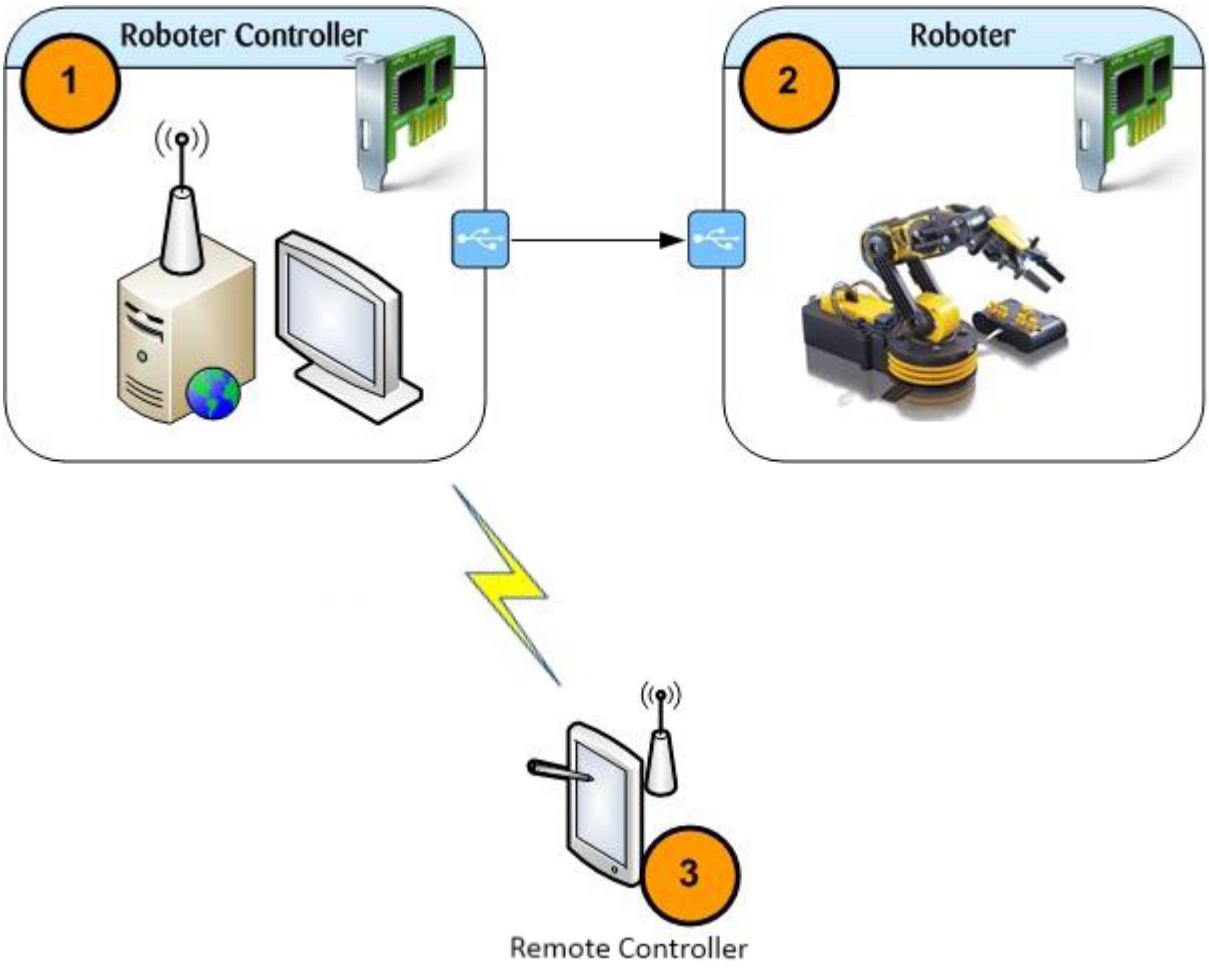


Abbildung 1: Projektscope

Der Roboter Controller ist die zentrale Steuereinheit für den Roboter, welcher auf einer Windows Embedded CE 6.0 R3 Plattform läuft. Der Roboter kann direkt am Roboter Controller über einen Touchscreen gesteuert werden. Des Weiteren bietet der Controller Services zur Steuerung des Roboters per Remote an. Der Remote Controller besteht aus einem GUI für die Fernsteuerung und einem Webservice Client für die Kommunikation mit dem Roboter Controller. Die Kommunikation zwischen Remote Controller und Roboter Controller basiert auf dem DPWS Standard, welcher in managed Code implementiert werden muss. Die Softwareentwicklung bezieht sich ausschliesslich auf den DPWS-Stack und die Software Applikationen für den Roboter Controller und den Remote Controller.

5.3 Roboter

Beim zu steuernden Roboter handelt es sich um einen Playtastic NC-1425. Dieser Roboter besitzt fünf Achsen(M1-M5) und eine LED (LED1). Das folgende Schema zeigt die vorhandenen Motoren und die Position der LED am Greifarm.

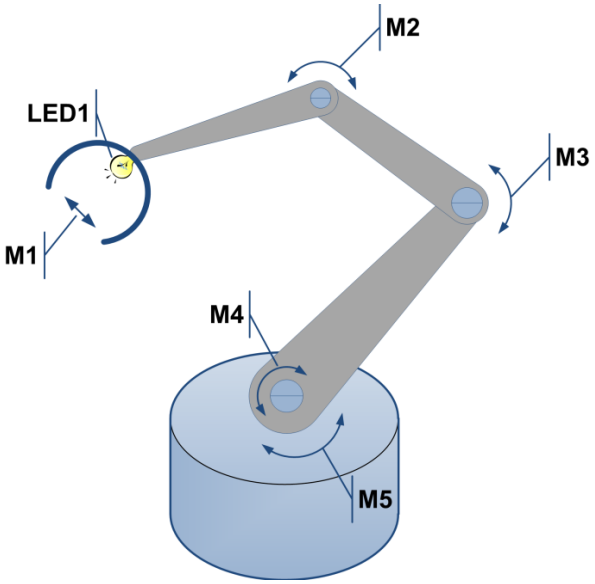


Abbildung 2: Roboter Schema

5.4 Referenzen

Folgende Dokumente wurden als Basis verwendet:

Referenz	Quelle
Aufgabenstellung Zühlke	/01_Aufgabenstellung/Beschreibung_Zühlke.pdf
Ergänzende Spezifikation	/01_Aufgabenstellung/HSR_Roboter_Controller_Specs.doc
Aufgabenstellung gemäss HSR	/01_Aufgabenstellung/Aufgabenstellung_HSR.pdf
Playtastic NC-1425 Manual	/90_Specifications/NC-1425_Manual_GERMAN.pdf

Tabelle 1: Referenzen

5.5 Übersicht

Die Anforderungsspezifikation ist in drei Komponenten aufgeteilt, wobei für jede Komponente alle Anforderungen separat beschrieben werden. Die drei Komponenten sind:

- DPWS-Stack
- Roboter Controller
- Remote Controller

6 Beschreibung

Zurzeit existiert keine managed Implementierung eines DPWS-Stacks (Device Profile for Web Services) für das Compact Framework. Innerhalb dieses Projektes soll unter anderem ein DPWS-Stack entwickelt werden, welcher in Zukunft von Zühlke für die Implementierung von Webservices in anderen Projekten verwenden kann.

Die Implementierung des DPWS-Stack wird in zwei Beispielapplikationen (Roboter Controller als Device, Remote Controller als Client) getestet. Die Schulungsplattform verwendet Zühlke intern für die Aus- und Weiterbildung zu technologiespezifischen Themen betreffend Windows CE und Windows Mobile.

Als Benutzer des DPWS-Stack sind ausschliesslich Software Entwickler vorgesehen, die im embedded und mobile Umfeld tätig sind. Teile der Gesamtapplikation sollen auch als Showcase für die Demonstration bei einem Kunden verwendet werden können.

7 Use Case Diagramm

7.1 Subsystem DPWS-Stack

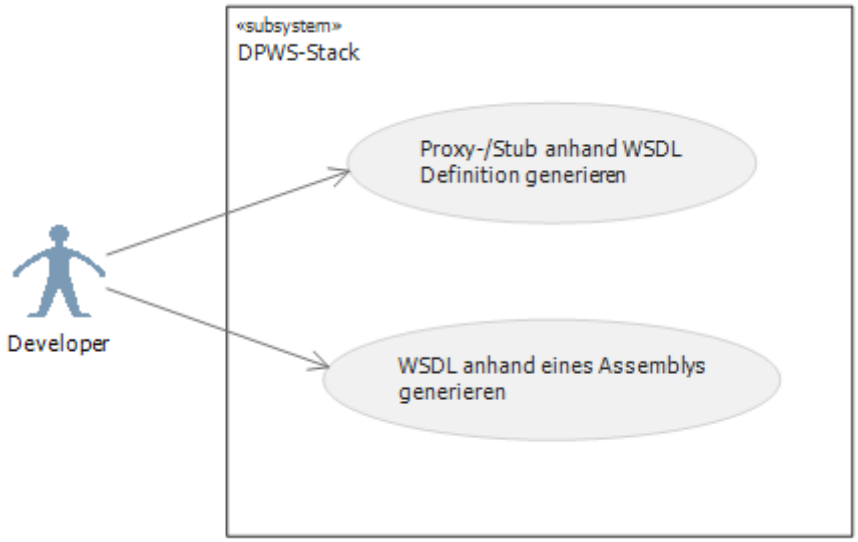


Abbildung 3: Use-Case Diagramm "Subsystem DPWS-Stack"

7.2 Subsystem Roboter Controller

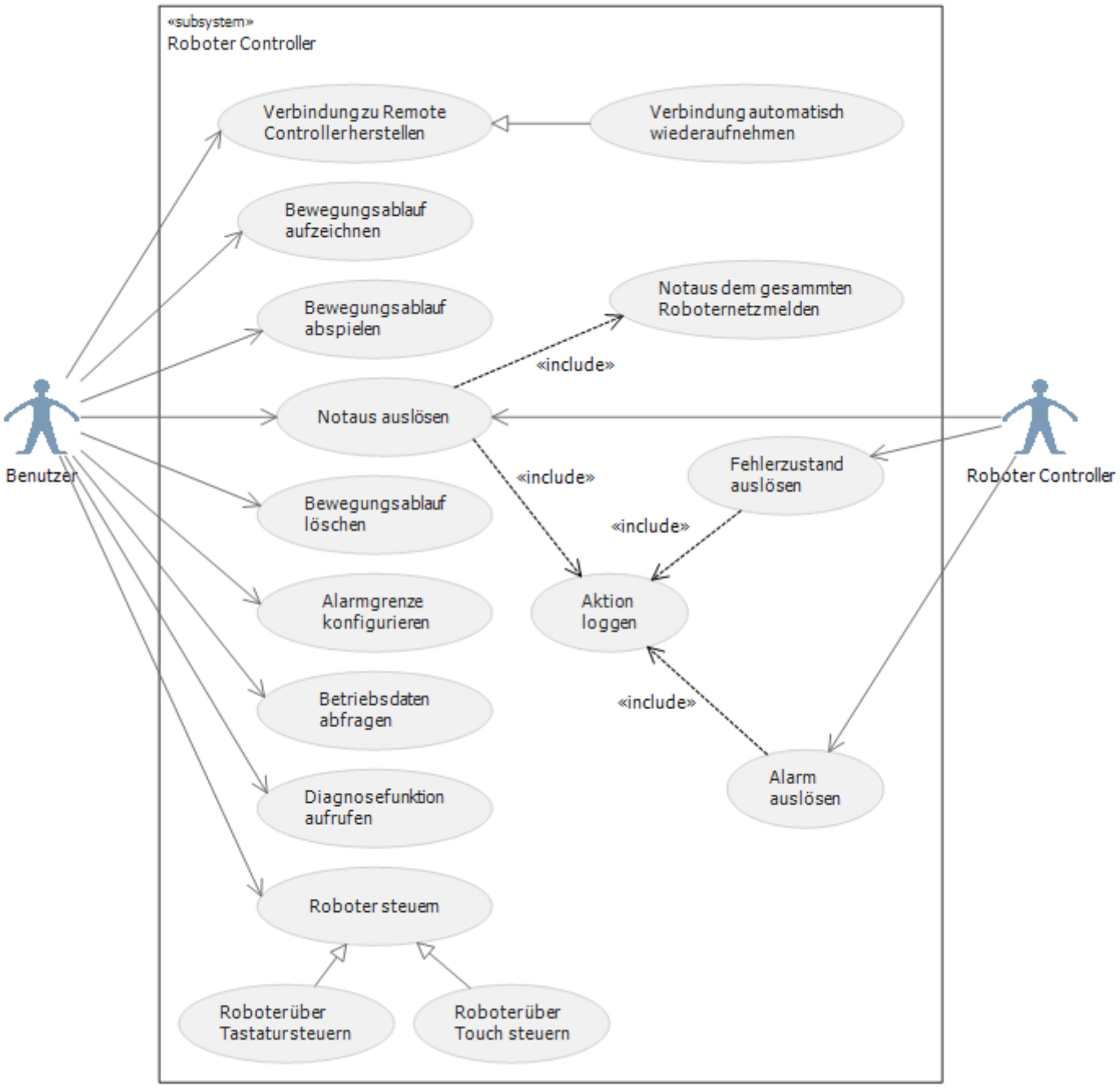


Abbildung 4: Use-Case Diagramm "Subsystem Roboter Controller"

7.3 Subsystem Remote Controller

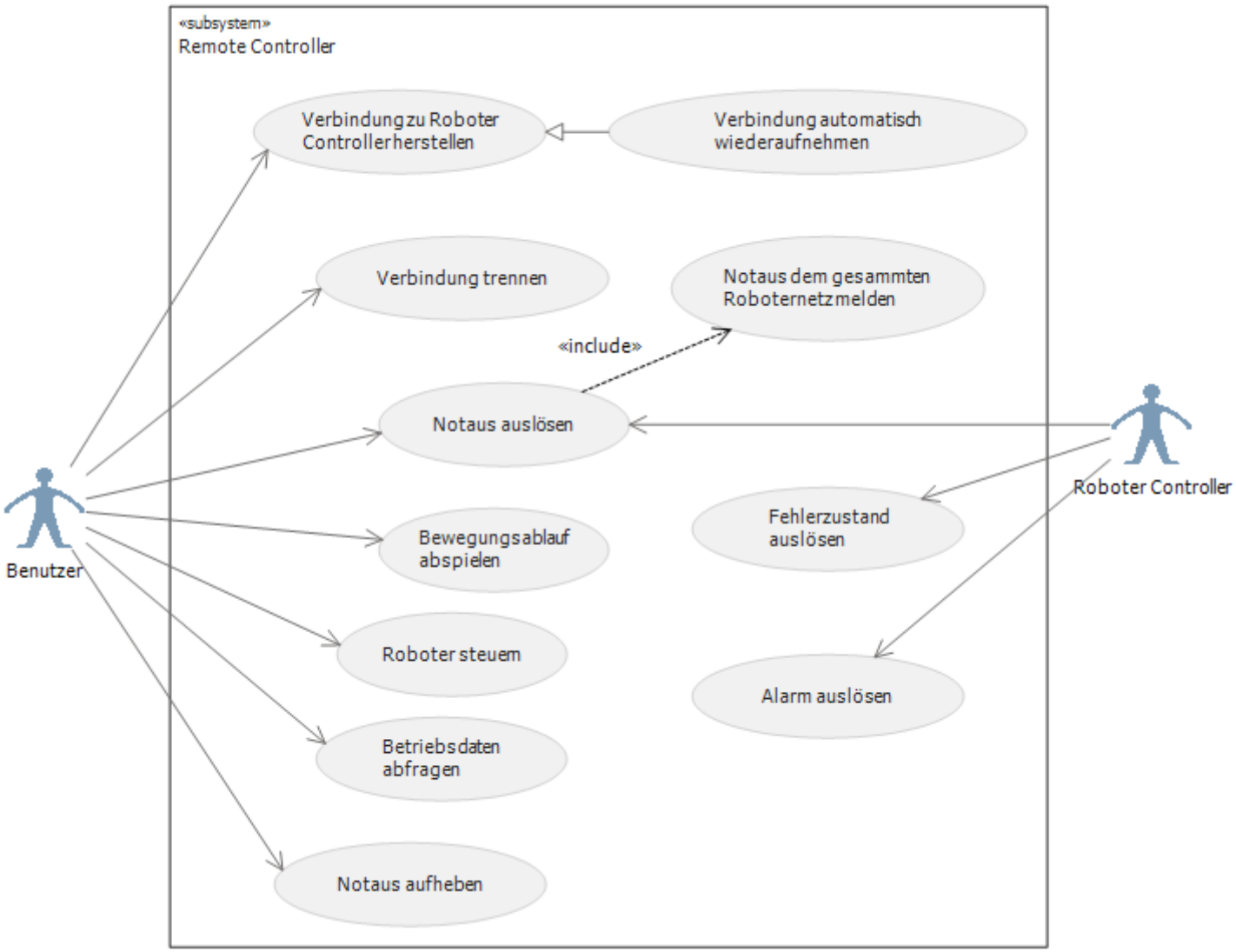


Abbildung 5: Use-Case Diagramm "Subsystem Remote Controller"

8 Allgemeine nicht funktionale Anforderungen

8.1 Qualität

NFR1	Qualität	Prio.	Typ
NFR1.1	Entwurfsrichtlinien	Hoch	Muss
Entwurfsrichtlinien gemäss FxCop einhalten. Es gelten die Standard Richtlinien von Microsoft.			
NFR1.2	UML-konforme Projektdokumentation	Hoch	Muss
UML Standards werden in allen Projektdokumentationen eingehalten.			
NFR1.3	Compilerfehler	Hoch	Muss
Keine Compilerfehler (Warning Level 4, 0 Warnings, 0 Errors)			
NFR1.4	MTBF	Hoch	Muss
Mean Time Between Failures: 30 Tage			

Tabelle 2: Allgemeine nicht funktionale Anforderungen zur Qualität

8.2 User Documentation

NFR2	User Documentation	Prio.	Typ
NFR2.1	API	Hoch	Muss
Dokumentation aller Interfaces und alle public API's (Methoden, Properties, etc.)			
NFR2.2	Windows Mobile 6.5 Tutorial	Hoch	Muss
Kleines Tutorial als Einstieg für die Applikationsentwicklung unter Windows Mobile 6.5			
NFR2.3	DPWS	Hoch	Muss
Entwicklerdokumentation für die Nutzung der DPWS Library			
NFR2.4	Bedienungsanleitung Remote Controller	Mittel	Soll
Bedienungsanleitung für den Remote Controller			
NFR2.5	Bedienungsanleitung Roboter Controller	Mittel	Soll
Bedienungsanleitung für den Roboter Controller			

Tabelle 3: User Documentations

9 DPWS-Stack

9.1 Systemfunktionen

9.1.1 DPWS Funktionalität

In dieser Bachelorarbeit müssen alle Komponenten des DPWS-Stack gemäss Spezifikation *[OASIS]* implementiert werden.

R1	DPWS Funktionalität	Prio.	Typ
R1.1	**Komponente Messaging**	**Hoch**	**Muss**
Die Messaging Komponente muss folgende Teile gemäss Standard implementieren: • URI • UDP • HTTP • SOAP 1.2 • WS-Addressing • Attachments			
R1.2	**Komponente Discovery**	**Hoch**	**Muss**
Die Komponente Discovery muss gemäss Standard implementiert werden.			
R1.3	**Komponente Description**	**Hoch**	**Muss**
Die Komponente Description muss gemäss Standard folgende Teile implementieren: • Characteristics • Hosting • WSDL • WS-Policy			
R1.4	**Komponente Eventing**	**Hoch**	**Muss**
Die Komponente Eventing muss gemäss Standard implementiert werden.			
R1.5	**Komponente Security**	**Mittel**	**Muss**
Die Komponente Security muss gemäss Standard implementiert werden.			

Tabelle 4: Funktionale Anforderungen zum DPWS-Stack

9.1.2 Visual Studio Integration

R2	Visual Studio Integration	Prio.	Typ
R2.1	**Proxy-/Stub Generierung anhand WSDL Definition**	**Mittel**	**Muss**
Es muss eine separate Komponente entwickelt werden, die anhand einer DPWS konformen WSDL Definition die passenden Proxy-/Stub Objekte generieren kann (analog Svcutil.exe für WCF).			

R2.2	WSDL Generierung anhand eines Assemblys	Mittel	Muss
Es muss eine separate Komponente entwickelt werden, die anhand eines Assemblys das passende WSDL generieren kann (analog Svcutil.exe für WCF).			
R2.3	Generierung innerhalb Visual Studio	Mittel	Soll
Die erstellte Komponente für das Generieren von Proxys und WSDL Files soll direkt innerhalb von Visual Studio verwendet werden können (analog WCF Integration).			

Tabelle 5: Funktionale Anforderungen zur Visual Studio Integration

9.1.3 MSBuild/TeamBuild Integration

R3	MSBuild/TeamBuild Integration	Prio.	Typ
R3.1	Integration des Code Generators in MSBuild/TeamBuild	Mittel	Muss
Die Generierung des Proxy-/Stub Codes bzw. WSDL Definition kann in MSBuild/TeamBuild integriert werden.			

Tabelle 6: Funktionale Anforderungen zur Visual Studio Integration

9.2 Schnittstellen

9.2.1 Software Schnittstellen

Die DPWS Implementierung muss in zwei wiederverwendbare Komponenten aufgeteilt werden. Eine Komponente für den Client und eine für den Server (Device).

9.2.2 Protokolle und Schnittstellen

Device Profile for Web Service ist ein Standard, welcher unter *[OASIS]*beschrieben ist.

9.3 Nicht funktionale Anforderungen

NFR3	Nicht funktionale Anforderungen DPWS	Prio.	Typ
NFR3.1	Design Constraints	Hoch	Muss
Die Schnittstellen zur Weiterverwendung des DPWS-Stacks müssen managed Code (.NET Compact Framework 3.5) sein.			

Tabelle 7: Nicht funktionale Anforderungen DPWS

10 Roboter Controller

10.1 Systemfunktionen

10.1.1 Robotersteuerung

R4	Steuern des Roboters	Prio.	Typ
R4.1	Roboterbefehle	Hoch	Muss
Es muss eine Robotersteuerung entwickelt werden, mit der alle Funktionen des Roboters Playtastic NC-1425 über den Roboter Controller gesteuert werden können. Es ist gewünscht, dass der Befehl so lange ausgeführt wird, wie die Taste gedrückt wird.			
R4.2	Steuerung über Touchinterface	Hoch	Muss
Der Roboter Controller besitzt ein Touchscreen, über den die Steuerbefehle an den Roboter gesendet werden können.			
R4.3	Steuerung über Tastatur	Mittel	Muss
Der Roboter muss über ebenfalls über eine optional anschliessbare Tastatur gesteuert werden können.			
R4.4	Roboter Controller agiert als Remote Controller	Mittel	Soll
Falls der Roboter Controller keinen Roboter per USB angeschlossen hat, kann dieser auch einen anderen Roboter per Webservice ansteuern.			

Tabelle 8: Funktionale Anforderungen zur Robotersteuerung

10.1.2 Bewegungsabläufe (Teaching)

Ein Bewegungsablauf setzt sich aus mehreren Befehlen zusammen, wobei zu jedem Befehl festgehalten wird, wie lange dieser ausgeführt werden soll.

R5	Bewegungsabläufe (Teaching)	Prio.	Typ
R5.1	Bewegungsablauf aufzeichnen	Hoch	Muss
Der Benutzer des Roboter Controllers kann beliebige Bewegungsabläufe aufzeichnen. Da der Roboter nicht programmatisch in einen Initialzustand gebracht werden kann, ist ein Bewegungsablauf stets unabhängig von einem Startpunkt. Der aufgezeichnete Bewegungsablauf muss mit einem Namen versehen und auf dem Roboter Controller permanent abgespeichert werden können.			
R5.2	Bewegungsablauf abspielen	Hoch	Muss
Auf dem Touchscreen des Roboter Controllers muss es möglich sein, die Liste aller gespeicherten Bewegungsabläufe anzuzeigen und einen bestimmten Ablauf abzuspielen.			
R5.3	Bewegungsablauf rückwärts abspielen	Mittel	Kann
Jeder Bewegungslauf soll auch rückwärts abgespielt werden können um einen Roboter nach einem Bewegungsablauf wieder in seine Startposition zu versetzen.			
R5.4	Bewegungsablauf löschen	Mittel	Soll
Der Benutzer des Touchinterfaces muss die Möglichkeit haben, alle gespeicherten Bewegungsabläufe darzustellen und davon beliebige Einträge zu löschen.			

Tabelle 9: Funktionale Anforderungen zum Teaching

10.1.3 Betriebsdaten

Als Betriebsdaten zählen die aufgelaufenen Betriebszeiten jedes einzelnen Motors. Für jede Betriebszeit kann eine individuelle Alarmgrenze gesetzt werden, siehe *10.1.8 Alarme*.

R6	Betriebsdaten	Prio.	Typ
R6.1	Betriebsdaten führen	Mittel	Muss
Da der Roboter selber keine Betriebsdaten führt, muss dies der Roboter Controller übernehmen. Der Roboter Controller muss für jeden einzelnen Motor die aufgelaufene Betriebszeit speichern.			
R6.2	Betriebsdaten über Schnittstelle verfügbar machen	Mittel	Muss
Der Roboter Controller muss die Betriebsdaten über eine Monitoringschnittstelle für Maintenance Aufgaben zur Verfügung stellen können.			

Tabelle 10: Funktionale Anforderungen zu den Betriebsdaten

10.1.4 Kopplung/Pairing mit einem Client

R7	Kopplung/Pairing mit einem Client	Prio.	Typ
R7.1	Schutz nach Geräteklassen	Hoch	Muss
Es soll ein Schutz nach Geräteklassen vorgenommen werden, damit der Roboter Controller nur auf Discovery Anfragen von gültigen Remote Controllern antwortet.			
R7.2	Verbindungsaufbau zu Roboter Controller	Hoch	Muss
Der Roboter Controller kann gleichzeitig nur von einem Remote Controller gesteuert werden. Sobald der Roboter Controller mit einem Remote Controller verbunden ist, darf dieser über das Discovery nicht mehr gefunden werden und keine neuen Verbindungen eingehen.			
R7.3	Automatische Wiederaufnahme der Verbindung	Niedrieg	Soll
Nach einem Verbindungsunterbruch, müssen Roboter Controller und Remote Controller die Verbindung automatisch wieder herstellen.			
R7.4	Auflösen des Pairing	Mittel	Muss
Das Pairing kann auf zwei verschiedene Arten aufgelöst werden: Entweder per Webservice oder, falls der „Partner“ momentan nicht erreichbar ist, manuell über das Touchinterface des Roboter Controllers. Sobald der Partner wieder verbindungsfähig ist, probiert dieser die Verbindung aufzubauen. Falls der Roboter besetzt ist, wird eine entsprechende Fehlermeldung angezeigt.			

Tabelle 11: Funktionale Anforderungen zur Kopplung/Pairing mit einem Client

10.1.5 Roboternetz

Ein Roboternetz definiert sich durch den Zusammenschluss mehrerer Roboter Controller in einem WLAN/LAN Segment.

Das Roboternetz ist in dieser Arbeit einzig für den gemeinsamen Notaus Zustand da. Innerhalb des Roboternetzes erfolgt keine andere Kommunikation.

R8	Roboternetz	Prio.	Typ
R8.1	Roboternetz aufbauen	Mittel	Muss
Sobald sich mehrere Roboter Controller im selben WLAN/LAN Segment befinden, so müssen sich diese automatisch zu einem Roboternetz zusammenschliessen.			
R8.2	Roboternetz verlassen	Mittel	Muss
Wird ein Roboter Controller heruntergefahren oder vom WLAN/LAN abgehängt, so müssen die anderen Roboter Controller im bestehenden Roboternetz dies erkennen.			

Tabelle 12: Funktionale Anforderungen zum Roboternetz

10.1.6 Notaus

Der Notaus Zustand bedeutet der sofortige Stopp im ganzen Roboternetz.

R9	Notaus	Prio.	Typ
R9.1	Notaus manuell auslösen	Hoch	Muss
Der Benutzer hat über den Touchscreen die Möglichkeit einen Notaus für den Roboter manuell auszulösen. Der Roboter muss in diesem Fall alle laufenden Bewegungsabläufe sofort stoppen.			
R9.2	Grenze für automatisches Notaus konfigurieren	Hoch	Muss
Über den Touchscreen muss dem Benutzer die Möglichkeit geboten werden, ein Timeout für die maximale ununterbrochene Tastenbetätigung zu setzen, welche als Grenze für ein automatisches Notaus gilt.			
R9.3	Automatischer Notaus auslösen	Hoch	Muss
Wird eine Taste länger als das gesetzte Timeout gedrückt, muss der Roboter Controller sofort ein automatischer Notaus auslösen.			
R9.4	Notaus dem Remote Controller weiterleiten	Hoch	Muss
Sobald ein Notaus ausgelöst wird, muss der Roboter Controller diese Meldung sofort dem Remote Controller weiterleiten, sofern er mit einem verbunden ist.			
R9.5	Notaus dem ganzen Roboternetz weiterleiten	Hoch	Muss
Sobald ein Notaus ausgelöst wird, muss der Roboter Controller diese Meldung sofort an alle anderen Roboter Controller im Roboternetz weitermelden, damit diese auch in den Notauszustand gelangen.			

R9.6	Notaus aufheben	Hoch	Muss
Der Benutzer kann das Notaus nur manuell über das Touchinterface am Roboter Controller aufheben. Ist der Roboter Controller Teil eines Roboternetzes, so hebt er immer automatisch den Notaus für alle anderen Roboter Controller auch auf.			

Tabelle 13: Funktionale Anforderungen zum Notaus

10.1.7 Fehlerzustände

Der Roboter Treiber kann pro Achse/Motor einen generellen Fehler werfen, welcher eine Fehlernummer (1 Byte) enthält. Zu jeder Fehlernummer ist ein Fehlertext zugeordnet.

R10	Fehlerzustände	Prio.	Typ
R10.1	Fehlerzustand anzeigen	Hoch	Muss
Der Roboter Controller muss jeden Fehlerzustand, der vom Treiber kommt, behandeln können und eine entsprechende Meldung auf dem Touchscreen anzeigen.			
R10.2	Fehlerzustände dem Remote Controller weiterleiten	Hoch	Muss
Der Roboter Controller muss jeden Fehlerzustand an den Remote Controller weiterleiten, sofern er mit einem verbunden ist.			

Tabelle 14: Funktionale Anforderungen zu den Fehlerzuständen

10.1.8 Alarme

Alarme werden ausgelöst, wenn eine bestimmte Alarmgrenze, welche vom Benutzer gesetzt wurde, überschritten wird. Diese Alarmgrenzen beziehen sich in dieser Arbeit ausschliesslich auf die Betriebsdaten des Roboters.

Der Roboter Controller führt die Betriebsdaten, da der Roboter diese selber nicht speichert.

R11	Alarme	Prio.	Typ
R11.1	Alarme konfigurieren	Mittel	Muss
Über das Touchinterface kann pro Achse/Motor eine Alarmgrenze gesetzt werden. Es wird definiert, wie lange eine Achse/Motor in Betrieb sein kann, bis ein Alarm ausgelöst wird. Der Benutzer muss die Möglichkeit erhalten individuell auszuwählen für welche Betriebsdaten die Überwachung aktiviert werden soll.			
R11.2	Alarme behandeln	Hoch	Muss
Wenn ein Alarm ausgelöst wird, muss der Roboter Controller den Alarm entsprechend behandeln, einen Service Eintrag schreiben und eine Service Meldung auf dem Touchscreen anzeigen.			
R11.3	Alarme dem Remote Controller weiterleiten	Hoch	Muss
Der Roboter Controller muss jeden Alarm an den Remote Controller weiterleiten, sofern er mit einem verbunden ist.			

R11.4	Alarmlevels setzen	Niedrig	Kann
Optional können pro Motor verschiedene Alarmgrenzen (Warning, Critical, Error) definiert werden.			

Tabelle 15: Funktionale Anforderungen zu den Alarmen

10.1.9 Logging

R12	Logging	Prio.	Typ
R12.1	Fehlerzustände loggen	Niedrig	Muss
Der Roboter Controller muss alle aufgetretenen Fehlerzustände loggen.			
R12.2	Alarme loggen	Niedrig	Muss
Der Roboter Controller muss alle aufgetretenen Alarme loggen.			
R12.3	Notaus loggen	Niedrig	Muss
Der Roboter Controller muss alle aufgetretenen Notaus Ereignisse loggen.			

Tabelle 16: Funktionale Anforderungen zum Logging

10.2 Schnittstellen

10.2.1 Benutzer Schnittstellen

Der Benutzer am Roboter Controller steuert den Roboter entweder über den Touchscreen oder über die Tastatur.

10.2.2 Hardware Schnittstellen

Die Kommunikation zwischen Roboter Controller und der Steuerplatine des Roboters erfolgt über eine USB Schnittstelle. Für die Kommunikation nach aussen enthält die Plattform ein WLAN Interface um darüber Webservices anzubieten. Zusätzlich erhält die Plattform einen LAN Anschluss, über den ein Zusammenschluss zu einem Roboternetz ermöglicht wird.

10.2.3 Software Schnittstellen

Das Steuern des Roboters wird als Webservice (DPWS) zur Remotesteuerung zur Verfügung gestellt. Über diese Schnittstelle kann folgendes angesteuert werden:

S1	Software Schnittstellen Roboter Controller	Prio.	Typ
S1.1	Abrufen der Betriebsdaten	Hoch	Muss
Der Roboter Controller muss alle Betriebsdaten über einen Web Service (DPWS) zur Verfügung stellen.			
S1.2	Bewegen der Motoren	Hoch	Muss
Der Roboter Controller muss einen Web Service (DPWS) zur Verfügung stellen, über den ein Remote Controller die Motoren des Roboters steuern kann.			
S1.3	Ein- bzw. ausschalten der LED	Hoch	Muss
Der Roboter Controller muss einen Web Service (DPWS) zur Verfügung stellen, über den ein Remote Controller die LED des Roboters ein- bzw. ausschalten kann.			
S1.4	Notaus auslösen/aufheben	Hoch	Muss

Der Roboter Controller muss einen Web Service (DPWS) zur Verfügung stellen, über den ein Remote Controller ein Notaus auslösen und wieder aufheben kann.

Tabelle 17: Software Schnittstellen Roboter Controller

10.2.4 Protokolle und Schnittstellen

Der Roboter Controller verwendet zur Kommunikation via Webservices den Server Teil des DPWS-Stacks *[OASIS]*.

10.3 Nicht funktionale Anforderungen

NFR4	Nicht funktionale Anforderungen Roboter Controller	Prio.	Typ
NFR4.1	**Performance**	**Hoch**	**Muss**
Da die Applikation Roboter Controller auf einem embedded System läuft, ist es wichtig, dass die Implementierung hinsichtlich Speicher und CPU Belastung optimiert wird.			
NFR4.2	**Usability**	**Hoch**	**Muss**
Das GUI auf dem Touchscreen muss modernen HMI Konzepten entsprechen und für einen Ingenieur konzipiert sein.			
NFR4.3	**Design Constraint**	**Hoch**	**Muss**
Das GUI auf dem Touchscreen soll mit Silverlight for embedded realisiert werden.			
NFR4.4	**Design Constraint I18n**	**Niedrig**	**Kann**
Umschaltung der Sprache für Roboter Controller und Remote Controller zw. Deutsch und Englisch sollte möglich sein.			

Tabelle 18: Nicht funktionale Anforderungen Roboter Controller

11 Remote Controller

11.1 Systemfunktionen

11.1.1 Robotersteuerung

R13	Robotersteuerung	Prio.	Typ
R13.1	Mit Roboter Controller verbinden	Hoch	Muss
Der Remote Controller muss die Möglichkeit haben mit einem beliebigen Roboter Controller eine Verbindung herzustellen.			
R13.2	Roboter steuern	Hoch	Muss
Sobald zwischen Remote Controller und Roboter Controller eine Verbindung besteht, kann der Roboter auf dieselbe Weise wie auf dem Touchscreen am Roboter Controller gesteuert werden.			
R13.3	Bewegungsabläufe starten	Mittel	Soll
Der Remote Controller erhält die Möglichkeit alle auf dem Roboter Controller gespeicherten Bewegungsabläufe anzuzeigen und davon beliebige Abläufe zu starten.			

Tabelle 19: Funktionale Anforderungen zur Robotersteuerung auf Remote Controller

11.1.2 Betriebsdaten

R14	Betriebsdaten	Prio.	Typ
R14.1	Betriebsdaten anzeigen	Niedrig	Muss
Der Benutzer kann über den Remote Controller alle Betriebsdaten des Roboters abfragen und diese entsprechend auf dem Display darstellen lassen.			

Tabelle 20: Funktionale Anforderungen zur Darstellung der Betriebsdaten auf Remote Controller

11.1.3 Fehlerzustände

R15	Fehlerzustände	Prio.	Typ
R15.1	Fehlerzustände anzeigen	Hoch	Muss
Der Remote Controller muss jeden Fehlerzustand, der vom Roboter Controller kommt, behandeln können und eine entsprechende Meldung auf dem Display anzeigen.			

Tabelle 21: Funktionale Anforderungen zu den Fehlerzuständen auf Remote Controller

11.1.4 Alarme

R16	Alarme	Prio.	Typ
R16.1	Alarme anzeigen	Hoch	Muss
Der Remote Controller muss jeden Alarm, der vom Roboter Controller kommt, behandeln können und eine entsprechende Meldung auf dem Display anzeigen.			

Tabelle 22: Funktionale Anforderungen zu den Alarmen auf Remote Controller

11.1.5 Notaus

R17	Notaus	Prio.	Typ
R17.1	Notaus anzeigen	Hoch	Muss
Der Remote Controller muss jedes Notaus, welches vom Roboter Controller kommt, bevorzugt (vor allen anderen Befehlen) behandeln und auf dem Display darstellen.			
R17.2	Notaus manuell auslösen	Hoch	Muss
Der Benutzer des Remote Controller muss die Möglichkeit haben den Notaus manuell über eine Taste am Display auszulösen.			
R17.3	Notaus auflösen	Hoch	Muss
Befindet sich der Roboter in einem Notaus Zustand, so muss dem Benutzer die Möglichkeit geboten werden, den Notaus manuell über eine Taste am Display aufzulösen. Ist der gesteuerte Roboter Controller Teil eines Roboternetzes, so hebt es automatisch den Notaus für alle anderen Roboter Controller auch auf.			

Tabelle 23: Funktionale Anforderungen zum Notaus auf Remote Controller

11.2 Schnittstellen

11.2.1 Benutzer Schnittstellen

Die Interaktion des Benutzers erfolgt über das Touchinerface des Windows Mobile Geräts.

11.2.2 Software Schnittstellen

Der Remote Controller konsumiert zur Steuerung des Roboters die Webservices des Roboter Controller.

11.2.3 Protokolle und Schnittstellen

Der Remote Controller verwendet zur Kommunikation via Webservices den Client Teil des DPWS-Stacks *[OASIS]*.

11.3 Nicht funktionale Anforderungen

NFR5	Nicht funktionale Anforderungen Remote Controller	Prio.	Typ
NFR5.1	Design Constraint	Hoch	Muss
Die Remote Controller Anwendung soll für Windows Mobile 6.5 implementiert sein.			

Tabelle 24: Nicht funktionale Anforderungen Remote Controller

Teil 3: Anhang

12 Verzeichnisse

12.1 Tabellenverzeichnis

12.2 Abbildungsverzeichnis

12.3 Literaturverzeichnis

[OASIS] http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html
März, 2010

Projektmanagement

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

1 Dokumenteninformationen

1.1 Zweck

Dieses Dokument beschreibt das ganze Projekt Management der Bachelorarbeit Roboter Controller.

1.2 Gültigkeitsbereich

Die Gültigkeit dieses Dokuments gilt für die gesamte Projektdauer.

1.3 Referenzen

Folgende Dokumente wurden als Basis verwendet:

Referenz	Quelle
Aufgabenstellung Zühlke	/03_Aufgabenstellung/Beschreibung_Zühlke.pdf
Ergänzende Spezifikation	/03_Aufgabenstellung/HSR_Roboter_Controller_Specs.doc
Aufgabenstellung gemäss HSR	/03_Aufgabenstellung/Aufgabenstellung_HSR.pdf

Tabelle 1: Referenzen

1.4 Übersicht

Das Dokument des Projekt Managements ist in vier Teile gegliedert. Zuerst wird im Projektplan die Projektorganisation, die Managementabläufe (Zeitplanung, Meilensteine, Iterationen) sowie das Risikomanagement dokumentiert. Der zweite Teil reflektiert über die durchlaufenen Iterationen. In den letzten beiden Teilen sind Erfahrungsberichte und Sitzungsprotokolle aufgeführt.

1.5 Änderungsgeschichte und Reviews

Revision				
Version	**Status**	**Datum**	**Beschreibung/Änderung**	**Autor**
0.1	Draft	01.03.2010	Formatvorlage erstellt	TZ
0.2	Draft	10.03.2010	Risiko Management Kapitel hinzugefügt	SZ
0.3	Draft	10.03.2010	Erster Entwurf der Projekt Organisation, Projektübersicht, Management Abläufe, Infrastruktur und Qualitätsmassnahmen geschrieben	SZ
0.4	Draft	11.03.2010	Überarbeitung des Risiko- & Buildmanagement, Iterationsplanung	TZ
0.5	Draft	11.03.2010	Risikomanagement nochmals überarbeitet	TZ
0.6	Draft	01.05.2010	Planung der Construction Iterationen und Releases dokumentiert	SZ
0.7	Draft	13.06.2010	Iterationsplanung bis und mit Construction 4	TZ

Tabelle 2: Änderungsgeschichte

Review		
Datum	**Kommentar**	**Revisor**
11.03.2010	Kleine Rechtschreibefehler korrigiert, Zeitberechnungen in Risiko Management korrigiert	SZ
10.06.2010	Review des ganzen Projektplans	TZ
15.06.2010	Review des ganzen Projektplans	SZ

Tabelle 3: Reviews

Teil 1: Projektplan

2 Projekt Übersicht

Im Rahmen dieser Bachelorarbeit sollen Teile einer Windows Embedded und Windows Mobile Schulungsplattform für die Zühlke Engineering AG entwickelt werden. Für den Aufbau dieser Schulungsplattform soll eine Steuerung (Roboter Controller) entwickelt werden, mit der ein Spielzeugroboter gesteuert werden kann. Der Roboter kann direkt über ein Touchscreen am Roboter Controller oder über ein Windows Mobile Gerät (Remote Controller) gesteuert werden. Die Kommunikation zwischen Remote Controller und Roboter Controller geschieht mittels einer eigenen Implementierung des DPWS-Stacks (Devices Profile for Web Services).

Die Schulungsplattform soll weiterhin zum Testen neuer GUI Technologien und Kommunikationskonzepten dienen.

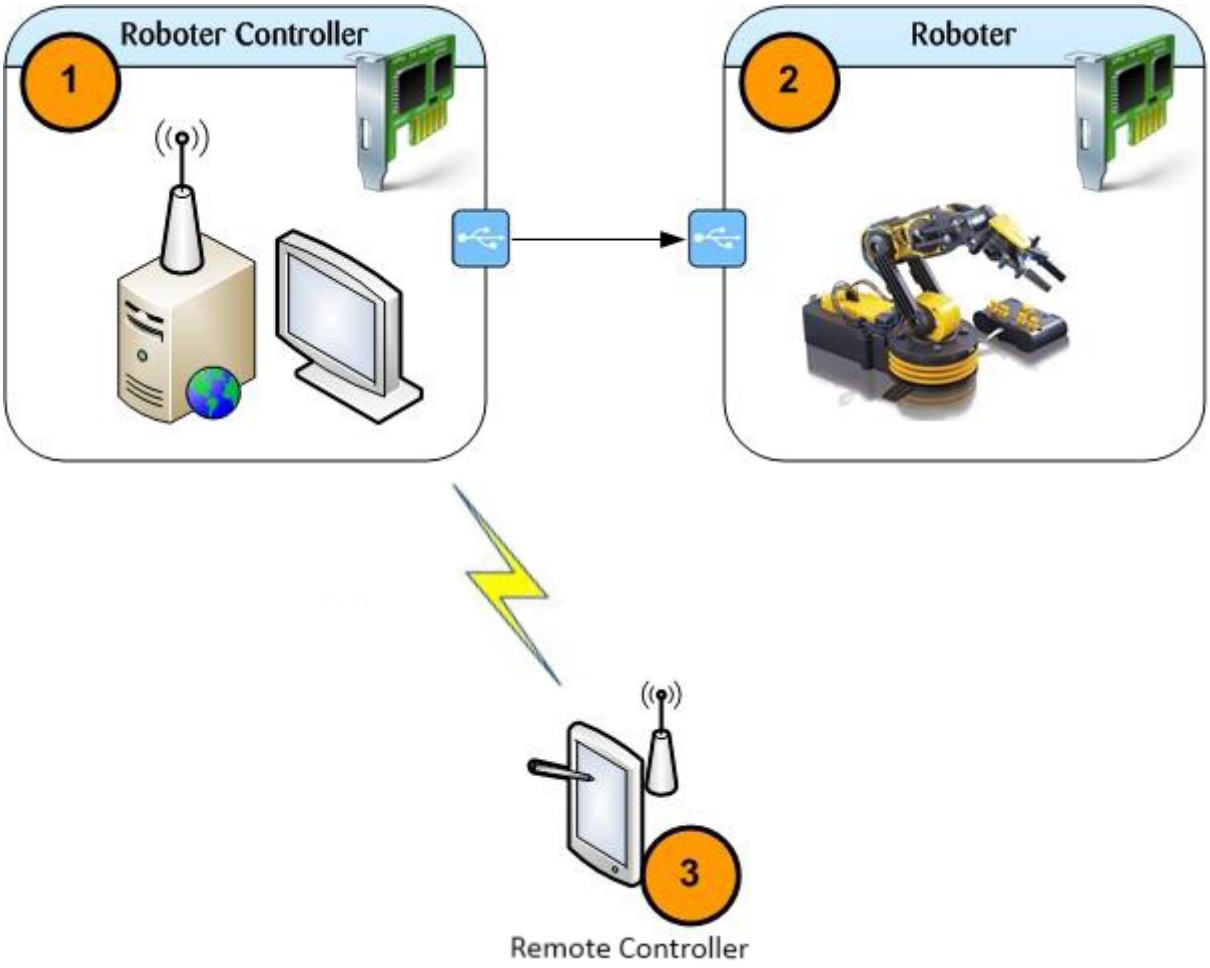


Abbildung 1: Systemarchitektur

Roboter Controller
Der Roboter Controller (1) besteht aus einer 32bit embedded Plattform mit einem ARM Controller. Als Betriebssystem wird Windows Embedded CE 6.0 R3 verwendet. Die Bedienung des Roboters über den Roboter Controller kann über ein Touchscreen oder die angeschlossene Tastatur geschehen. Das GUI auf dem Touchscreen soll modernen HMI Konzepten entsprechen und in Silverlight für embedded Systeme umgesetzt werden.

Remote Controller
Der Remote Controller (3) ist ein Handy mit dem Betriebssystem Windows Mobile 6.5. Über den Remote Controller kann der Roboter ferngesteuert werden. Ein Remote Controller ist

dafür immer mit einem Roboter Controller verbunden und kommuniziert über Web Services.

3 Projektorganisation

Die folgende Tabelle listet alle Projektbeteiligten und ihre Funktionen im Projekt auf.

Projektbeteiligter	Funktion
Tobias Zürcher (TZ)	BSc Student
Stefan Züger (SZ)	BSc Student
Prof. Hansjörg Huser (HH)	Betreuer, HSR
Dipl. Ing. Jürg Jucker (JJ)	Betreuer Stellvertretung, HSR
Dipl. Ing. Mario Klessascheck (MKL)	Auftraggeber, Zühkle Engineering AG
Dipl. Ing. Michale Koster (MKO)	Technischer Berater, Zühlke Engineering AG
Prof. Beat Stettler	Gegenleser Bachelorarbeit, HSR
Stefan Zettel	Experte Bachelorarbeit, Ascentiv AG

Tabelle 4: Projektbeteiligte

4 Management Abläufe

4.1 Projekt Zeitaufwand

Der Umfang für dieses Projekt beträgt pro Person 360 Stunden. Mit zwei Studenten liegt somit der Gesamtumfang für die ganze Bachelorarbeit bei 720 Stunden

Der Start der Bachelorarbeit ist am 22. Februar 2010 und dauert 16 Wochen bis zum 18.06.2010. Während den 16 Wochen teilt sich der wöchentliche Arbeitsaufwand so auf, dass die ersten 14 Wochen pro Person 20 Stunden und in der Woche 15 und 16 jeweils 40 Stunden aufgewendet werden. Zwischen Woche 6 und 7 liegt eine Woche Osterferien.

4.2 Projektplan

4.2.1 Meilensteine

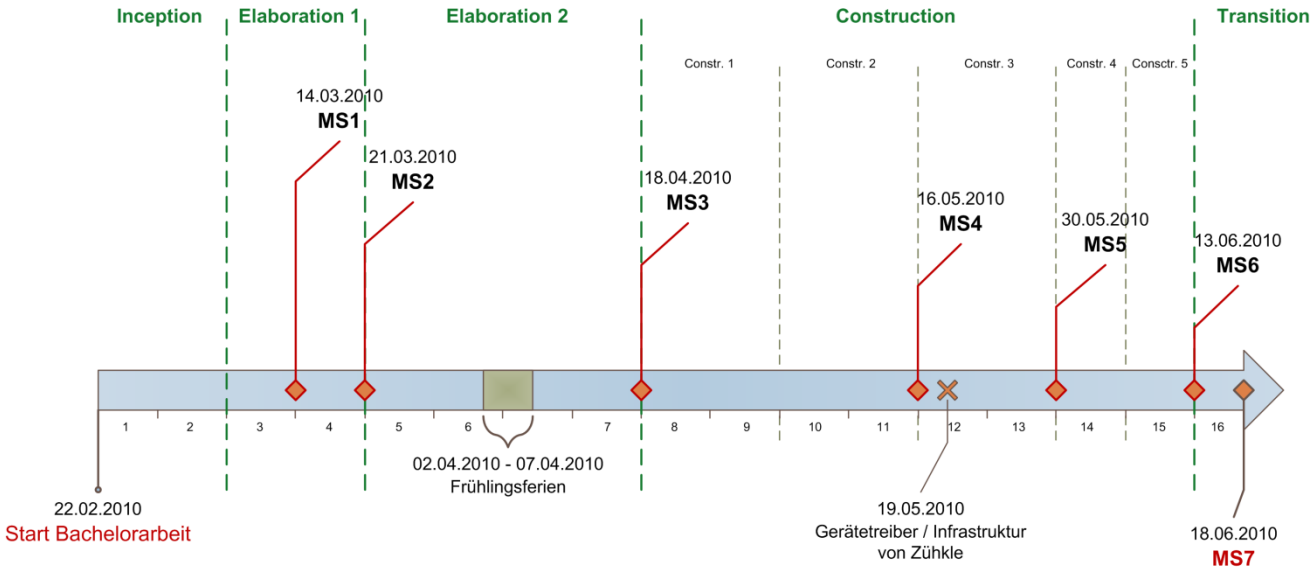


Abbildung 2: Meilensteine

In der folgenden Tabelle werden die einzelnen Meilensteine grob beschrieben.

MS	Bezeichnung	Beschreibung	Woche
MS1	Projektplan Review	Projektplan bis und mit Elaboration Phase fertig gestellt, Review mit Betreuer	W03
MS2	Anforderungen und Analyse	Erste Iteration Elaboration abgeschlossen: Software Requirements Specification abgeschlossen und von Auftraggeber bestätigt.	W04
MS3	Ende Elaboration	Zweite Iteration Elaboration abgeschlossen: Architekturprototyp über alle Schichten, Artefakte der OOD abgeschlossen, SAD grösstenteils fertiggestellt	W07
MS4	Alpha-Version	Verbindungsaufbau zu Roboter Controller möglich, Roboter Steuerung möglich per Remote & Touch	W11
MS5	Beta-Version	Notaus inklusive Weiterleitung ins Roboternetzwerk, Alarmmeldungen, Bewegungsabläufe aufnehmen & abspielen	W13
MS6	Ende Construction	Gesamte Funktionalität implementiert, Systemtests durchgeführt	W15
MS7	Ende Bachelorarbeit	Abgabe der Bachelorarbeit	W16

Tabelle 5: Meilensteinübersicht

4.2.2 Iterationsplanung

Der Rational Unified Process definiert vier Phasen. Diese vier Phasen werden in mehreren Iterationen durchlaufen. Die folgende Tabelle zeigt eine Übersicht über die einzelnen Iterationen und beschreibt diese mit einigen Stichworten.

Iteration	Beschreibung	Ende	Dauer [W]
Inception 1	Kickoff-Meeting, Kennenlernen aller Projektbeteiligten, Risiko Management Plan, Formatvorlagen, Beginn Projektplan, Entwicklungsumgebung aufsetzen, Team Foundation Server 2010 evaluieren und Interview mit Auftraggeber	W02	2
Elaboration 1	Use Case Model, Projektplan erweitern, Domain Modell, Software Requirements Specification, Technologiestudium	W04	2
Elaboration 2	Design Modell, Logische Architektur der Packages, Projektplan erweitern, Architekturprototyp, Demonstration des Prototyps bei Zühlke	W07	3
Construction 1	Aufbauen der Software-Architektur, GUI Entwurf Roboter Controller, GUI Entwurf Remote Controller, Definition Service Schnittstelle	W09	2
Construction 2	Implementierung der Verbindung zwischen Remote und Roboter Controller, Robotersteuerung mit Dummy-Treiber, Betriebsdaten führen und anzeigen, Fehlerzustände anzeigen	W11	2
Construction 3	Implementierung der Notaus und Alarmfunktionen, Bewegungsabläufe aufzeichnen und abspielen	W13	2
Construction 4	Integration in Visual Studio, Code Generatoren	W14	2
Construction 5	Reserviert für Bugfixing und Fertigstellen der offen gebliebenen Features	W15	1
Transition	Poster, Management Summary, Abstract, Qualitätssicherung, Benutzerdokumentation, Vorbereitung der Präsentation	W16	1

Tabelle 6: Übersicht der Iterationen

In der Elaboration Phase werden die technischen Risiken beseitigt. Als „proof of concept“ wird ein Architekturprototyp entwickelt, welcher über alle Layers (DPWS als Kommunikation, Silverlight for Embedded als UI Layer) funktioniert. Anschliessend wird in der Construction der Remote- bzw. Roboter Controller implementiert. Die Organisation der einzelnen Iterationen innerhalb der Construction Phase ist in mehrere kurze Construction Iterationen aufgeteilt.

4.2.3 Besprechungen

Es findet ein wöchentliches Meeting mit den Betreuern an der HSR statt um allfällige Fragen zu klären, den Projektfortschritt zu besprechen oder Reviews durchzuführen.

Teilnehmer TZ, SZ, JJ, HH
Ort HSR Rapperswil Gebäude 6, Zimmer 6.010

4.2.4 Releases

Folgende Tabelle zeigt den Umfang jedes Releases.

Typ	Woche	Beschreibung
Prototyp	7	• Proof-of-concept Anwendung über alle Layer • One-Way, Two-Way und Eventing Kommunikation über DPWS • UI auf embedded System mit Silverlight for Embedded inkl. Animationen
Alpha	11	• Verbindungsauf- bzw. abbau zwischen Remote und Roboter Controller • Steuerung des Roboters gegen einen Dummy-Treiber über Remote und Roboter Controller • Führen der Betriebsdaten • Roboternetz aufbauen / verlassen
Beta	13	• Bewegungsabläufe aufzeichnen, abspielen, löschen • Alarm und Notaus Features • Roboter-LAN (Notaus werden weitergleitet)
Final	15	• Alle spezifizierten Features umgesetzt

Tabelle 7: Übersicht der Releases

5 Risiko Management

Das folgende Kapitel dient der Analyse der potentiellen Projektrisiken und deren Vorbeugung. Es wird analysiert, welche Auswirkungen ein Eintreten eines Risikos mit sich bringt und mit welchen Massnahmen diese Risiken eingedämmt werden können.

5.1 Risiko Liste

ID	Bezeichnung	Risikobeschreibung	Auswirkungen	EW	SP [h]	K [h]
R01	Requirements	Die Anforderungen sind über lange Zeit unklar oder werden vom Projektteam anders interpretiert, als es der Kunde möchte.	Endresultat entspricht nicht den Wünschen des Kunden.	5%	30	2
R02	TFS 2010	Es stellt sich heraus, dass der TFS 2010 nicht geeignet für dieses Projekt ist.	Ineffizientes Projektmanagement	10%	20	2
R03	Verfügbarkeit der Infrastruktur	Die gesamte Infrastruktur mit Embedded Hardware und Roboter ist zu spät verfügbar.	Systemtests mit der effektiven Infrastruktur kommen zu kurz. Qualität des Endprodukts kann nicht sichergestellt werden. Es können nicht alle Features rechtzeitig umgesetzt werden.	5%	20	1
R04	Roboter Treiber	Der Roboter Treiber wird von Zühlke nicht pünktlich fertiggestellt.	Der Roboter kann nicht im System eingebunden werden.	5%	16	1
R05	DPWS-Stack	DPWS Implementierung ist umfangreicher als geplant.	Weniger Zeit für die Implementation der Showcase Applikation	20%	40	8
R06	DPWS-Stack, TCP/IP	TCP/IP Probleme vom Compact Framework für die Implementierung des DPWS-Stacks.	Abhängige Komponenten können nicht getestet werden, Integrationstests können nicht rechtzeitig durchgeführt werden.	5%	20	1
R07	Remote Controller: Silverlight for embedded	Silverlight for Embedded ist nicht ausgereift, Funktionen fehlen → Implementierung mit C++ umständlich	GUI wird weniger modern aussehen, weniger oder keine Animationen.	30%	20	6
R08	Remote Controller	Testumgebung ist nicht emulierbar, was zu einer erhöhten Entwicklungszeit führt.	Es wird weniger getestet. Gefahr grösser, das Bugs erst im Systemtest erkennt werden.	10%	20	2

R09	Integration in Visual Studio	Code für Generierung der Proxy/Stubs muss komplett neu geschrieben werden.	Da VS Integration gegen Ende des Projektes realisiert wird, kann die Zeit knapp werden, so dass diese Funktionalität nicht fertiggestellt werden kann.	20%	10	2
R10	Undefinierte Treiberschnittstelle	Die API vom Robotertreiber ist nicht im Voraus definiert.	Treiberschnittstelle kann nicht im Voraus getestet werden. Da Treiber spät geliefert wird bleibt eventuell zu wenig Zeit um sauber zu testen.	5%	10	1
Total einzuplanende Reserven in Arbeitsstunden [h]						**26**

Tabelle 8: Risiko Management Liste

Legende: EW Eintrittswahrscheinlichkeit
SP Schadenspotential (in Arbeitsstunden h)
R Einzuplanende Reserve (EW * SP, in Arbeitsstunden h)

5.2 Massnahmen zur Risikovermeidung

Die folgende Tabelle zeigt, welche konkreten Massnahmen im Bezug auf die einzelnen Risiken vorgenommen werden. Falls möglich, wird zusätzlich angegeben, bis zu welchem Datum die getroffenen Massnahmen getroffen werden müssen.

Mit einem umfangreichen Architekturprototyp am Ende der Elaboration 2 sollen so gut wie alle Risiken zum grössten Teil minimiert worden sein.

	Bezeichnung	Massnahmen	Datum
R01	Requirements	Die Software Requirements Specification wird so detailliert ausformuliert wie möglich Zühlke zum Review gegeben. Zühlke hat die Möglichkeit Unstimmigkeiten zu melden. Sobald die Requirements komplett ausgearbeitet sind, muss der Auftraggeber diese bestätigen.	21.03.2010 MS2
R02	TFS 2010	In der Inception Phase wird das Setup eines neuen Team Foundation Servers 2010 intensiv getestet. Am Ende der Inception wird entschieden, ob der TFS weiterhin verwendet wird oder ob allenfalls trotzdem ein SVN Server eingesetzt werden soll.	07.03.2010 Ende Inception
R03	Verfügbarkeit der Infrastruktur	Es wird mit Zühlke ein Termin ausgehandelt, an dem die komplette Infrastruktur an der HSR verfügbar sein muss, damit das Projekt rechtzeitig abgeschlossen werden kann. Falls der Termin von Zühlke nicht eingehalten wird, ist ein Fallback geplant, indem man mit dem integrierten Windows Embedded Emulator vom Visual Studio testet und die Roboter Funktionalität mit einem eigenen GUI simuliert.	19.05.2010
R04	Roboter Treiber	Es wird mit Zühlke ein Termin abgemacht, an dem der Roboter Treiber verfügbar sein muss. Als Fallback wird bereits für den Architekturprototyp ein kleiner Roboter-Simulator gebaut, der die Roboter Befehle auf der Konsole ausgibt.	19.05.2010

R05	DPWS-Stack	Mit einem Architekturprotoyp sollen alle kritischen Teile des DPWS-Stacks bereits abgedeckt werden. Falls die Zeit trotzdem nicht ausreicht, muss mit dem Kunden abgesprochen werden, auf welche Komponenten des DPWS-Stacks verzichtet werden könnte.	18.04.2010 MS3
R06	DPWS-Stack, TCP/IP	Frühzeitig Tests ausserhalb der emulierten Testumgebung tätigen, um Probleme möglichst früh zu erkennen.	18.04.2014 MS3
R07	Silverlight for embedded	Mit dem Architekturprotoyp sollen alle kritischen Teile von Silverlight for Embedded bereits abgedeckt werden. Die Fallbacklösung ist ein Windows Forms GUI.	18.04.2010 MS3
R08	Remote Controller	Mit dem Architekturprotoyp sollen alle kritischen Teile des Remote Controllers bereits abgedeckt werden.	18.04.2010 MS3
R09	Integration in Visual Studio	Frühzeitig überprüfen, ob der Quellcode der Hilfstools von Microsoft (MFSvcUtil.exe) verfügbar ist.	18.04.2010 MS3
R10	Undefinierte Treiberschnittstelle	Definition der Treiberschnittstelle frühzeitig mit Zühlke klären (Elaboration 2). Die Tests werden mit einer Dummy-Implementation ausgeführt. Sobald der Treiber verfügbar ist, kann die Dummy-Implementation mit dem richtigen Treiber ersetzt werden.	18.04.2010 MS3

Tabelle 9: Massnahmen zur Risikovermeidung

5.3 Risiko Zusammenfassung

Wie in der Risiko Liste im Kapitel *5.1 Risiko Liste* ermittelt wurde, beträgt das gewichtete Schadenspotential aller Risiken für das Projekt *26 Arbeitsstunden*!

Bei einer wöchentlichen Arbeitszeit von 20 Stunden pro Person, entspricht dies weniger als einer Arbeitswoche. Auf die 16 Projektwochen verteilt ist dieser Anteil genügend gering, dass diese Reservezeit durch einen Wochenendeinsatz kompensiert werden könnte.

6 Infrastruktur

6.1 Hardware

Gerät	Beschreibung
Persönliches Notebook	Für alle Arbeiten an der Bachelorarbeit von zu Hause
Desktop PC im Bachelorarbeits-zimmer	Für alle Arbeiten an der Bachelorarbeit an der HSR in Rapperswil
HTC HD2	Handy für die Entwicklung und das Testen des Remote Controllers
Playtastic NC-1424	Roboter, der vom Roboter Controller gesteuert wird
Roboter Controller Hardware	Toradex Board
WLAN-Router	Router für das BA-RobotControl LAN
WLAN-Dongles	USB WLAN-Dongles damit Workstations eine Verbindung zum BA-RobotControl LAN herstellen können.

Tabelle 10: Liste verwendeter Hardware

6.2 Software

Produkt	Beschreibung
.NET Compact Framework 3.5	Entwicklung der Anwendung auf dem Windows Mobile Gerät und des DPWS-Stacks
Enterprise Architect	Erstellen von Software Diagrammen
Expression Blend 2	Design von GUIs für Silverlight for embedded
Microsoft Office 2007	Dokumentation
Microsoft Visio 2007	Erstellen von Diagrammen
NClass	Gratis Programm für das Erstellen von UML Diagrammen im Visual Studio look & feel.
Team Foundation Server 2010 RC	Projekt-, Versions-, Dokumenten- und Buildmanagement
Visual Studio 2005	Gebraucht für die Verwendung des Platform Builders für Windows Embedded CE
Visual Studio 2008	Primäre Entwicklungsumgebung für alle Programmiertätigkeiten
Visual Studio 2010 RC	Für Projektmanagement und neue TFS 2010 Features
Windows 7	Betriebssystem auf Entwickler-PCs
Windows Embedded CE 6.0 R3	Betriebssystem auf embedded Sytem
Windows Server 2008	Betriebssystem auf dem Server, wo TFS 2010 installiert ist

Tabelle 11: Liste verwendeter Software

6.3 Team Foundation Server

In dieser Bachelorarbeit wird mit dem Team Foundation Server 2010 RC gearbeitet. Folgende Punkte werden in diesem Projekt vom TFS gehandhabt:

- Projektmanagement (Tasks, User-Stories, Zeiterfassung, Reports)
- Sourcecode Verwaltung
- Dokumentenverwaltung (Sharepoint)
- Bug & Issue Tracking
- Buildmanagement
- Wiki
- Team Kalender

Das Team Portal vom TFS ist unter folgender URL erreichbar: http://ba.codevision.ch/.

7 Qualitätsmassnahmen

7.1 Teammeetings / Sitzungsprotokolle

Wie im Kapitel *4.2.3 Besprechungen* beschrieben, organisieren wir wöchentliche Betreuer Meetings. Für jedes Meeting muss ein Sitzungsprotokoll geschrieben werden, in dem alle Erkenntnisse und Entscheide aus dem Meeting sowie alle neu definierten Tasks protokolliert sind. Die Sitzungsprotokolle sind im Ordner *05_Projektmanagement\Sitzungsprotokolle* zu finden.

7.2 Testing

7.2.1 Unit-Tests

Jeder Entwickler ist aufgefordert an sinnvollen Stellen in seinem Source-Code fortlaufend Unit-Tests zu schreiben und gegen diese Tests zu entwickeln. Bevor ein Entwickler seinen neuen Source Code ins Repository eincheckt, müssen alle seine Unit-Tests erfolgreich absolviert worden sein.

7.2.2 System-Tests

Nach den Unit-Tests werden in einer separaten Testphase verschiedene System-Tests durchgeführt und dokumentiert. Die Testdokumentation ist im Ordner *06_Test* zu finden.

7.3 Reviews

7.3.1 Code-Reviews

Jeder Code muss mindestens einmal durch eine andere Person geprüft und wenn möglich verbessert werden. Änderungen werden mündlich kommuniziert und müssen nicht im Code dokumentiert werden.

Der Revisor überprüft:

- Verständlichkeit des Codes (evtl. durch Kommentare ergänzen)
- Einhalten der Codierungsrichtlinien unter Mithilfe des „Microsoft StyleCop“
- Korrektheit des Codes
- Auftreten von „Code Smells“
- Effizienz von komplexen Algorithmen

7.3.2 Dokumenten-Reviews

Jedes geschriebene Dokument muss mindestens einmal von einer anderen Person durchgeschaut und wenn nötig verbessert werden. Der Revisor trägt sich in die „Revision History“ des entsprechenden Dokuments mit Datum und Kommentar ein.

Der Revisor überprüft:

- Rechtschreibung / Grammatik
- Einhalten des vorgegebenen Dokumenten Templates
- Konsistenz der verschiedenen Texte
- Vollständigkeit des Dokuments

7.3.3 Dokumentenvorlage

Damit alle geschriebenen Dokumente einen einheitlichen Stil befolgen, gibt es eine Word Formatvorlage an folgender Stelle: *00_Templates\ba_blue-grey_template.dotx*. Jedes neu erstellte Dokument muss mit diesem Template erstellt und die vordefinierten Formatvorlagen verwendet werden.

7.3.4 Codierungsrichtlinien

Um sicherzustellen, dass jeder Programmcode, unabhängig vom Autor, einheitlich aussieht, gelten für das Projekt die allgemeinen .NET Codierungsrichtlinien von Microsoft. Diese sind auf der folgenden MSDN Seite ausführlich beschrieben:

http://msdn.microsoft.com/en-us/library/ms229042.aspx

Jedes Projektmitglied ist verpflichtet, diese Codierungsrichtlinien genau studiert zu haben und auch einzuhalten.

Um die Einhaltung der Richtlinien effizient zu überprüfen, wird Microsoft StyleCop verwendet. Dieses Tool bietet eine automatisierte Überprüfung der Einhaltung aller definierten Guidelines.

8 Zeitplanung

Gemäss Vorgabe müssen pro ECTS Punkt 30 Arbeitsstunden geleistet werden. Die Bachelorarbeit wird mit 12 ECTS Punkten belohnt, somit müssen 360 Stunden pro Student aufgewendet werden.

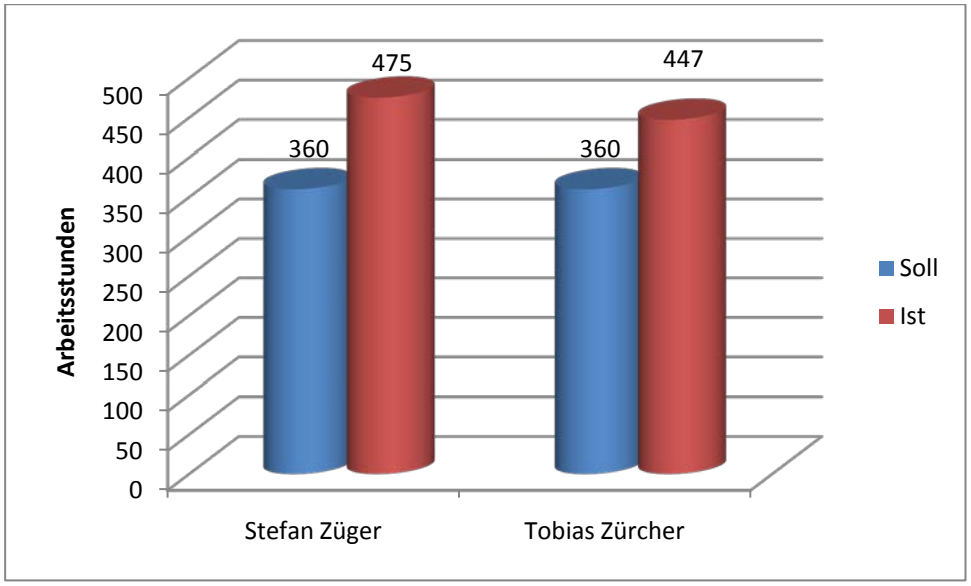


Abbildung 3: Arbeitsstunden total

Da für die Bachelorarbeit 16 Wochen (Woche 7 ist Ferien) zur Verfügung stehen, muss jeder Student in den ersten 14 Arbeitswochen jeweils 20 Stunden und in den letzten beiden Wochen 40 Stunden arbeiten.

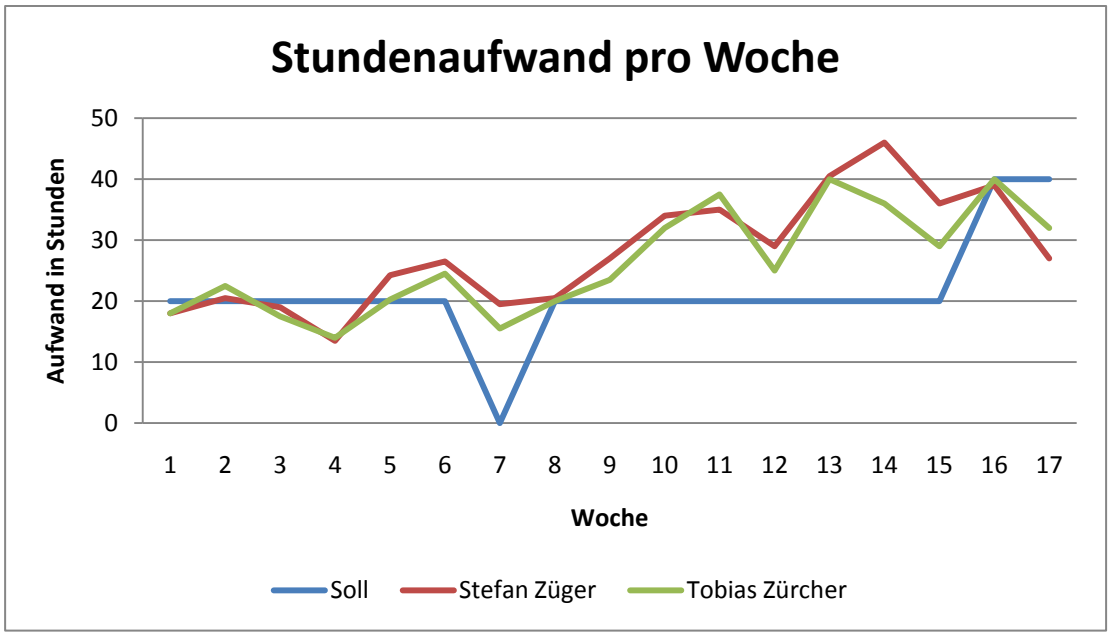


Abbildung 4: Stundenaufwand pro Woche

8.1 Arbeitsaufteilung

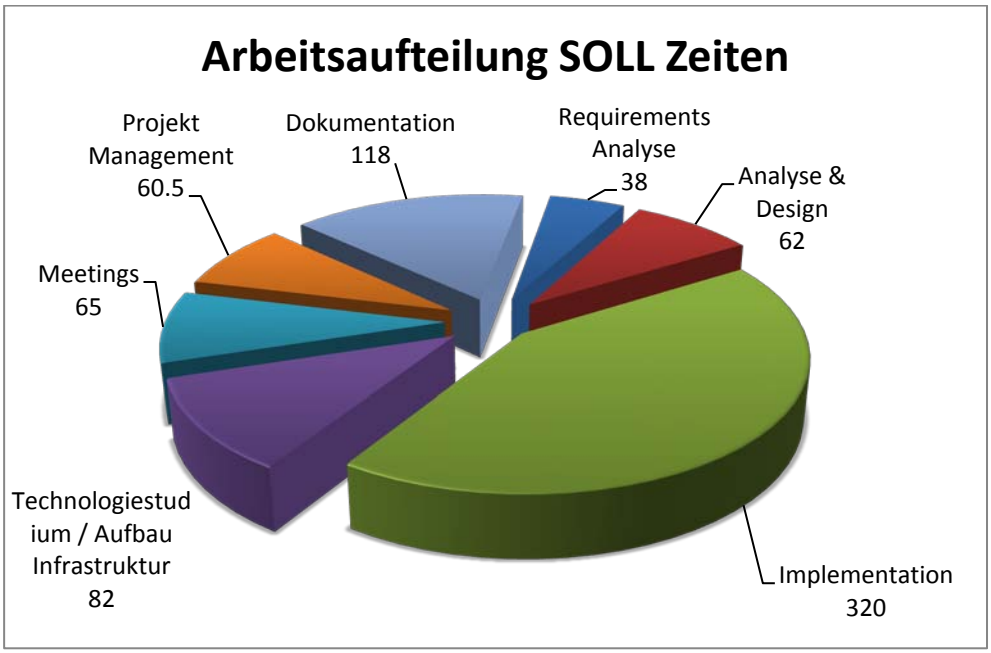


Abbildung 5: Arbeitsaufteilung Soll Zeiten

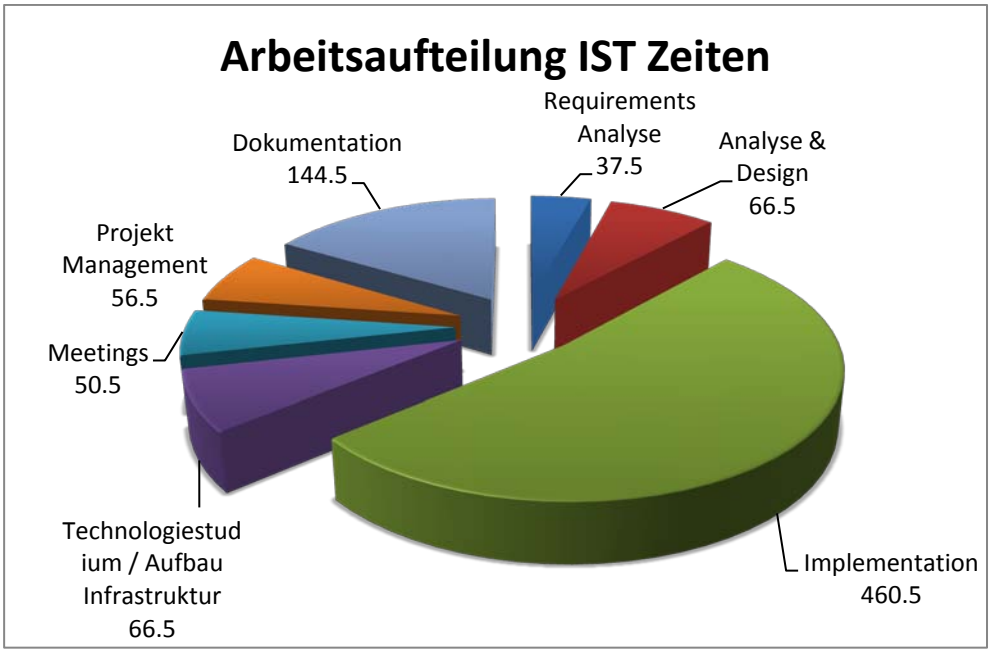


Abbildung 6: Arbeitsaufteilung Ist Zeiten

Das Testen wurde nicht explizit auf eine separate Kategorie verbucht. Unit-Tests wurden während der Implementation erstellt und sind darum in der Kategorie Implementation verbucht. Systemtests sind der Kategorie Dokumentation zugeordnet.

8.2 Aufwandvergleich der User Storys

Beim Vergleich der Soll und Ist Zeiten der 20 aufwändigsten User Storys fällt auf, dass die Umsetzung beinahe aller User Storys länger dauerte als geplant. Gründe dafür waren:

- Probleme bei Fehlersuche und Behebung der Memory Leaks bei Kommunikation zwischen managed und native Code
- Liebe zum Detail (Gestaltung der UIs, UI Animationen etc.), die nicht unbedingt gefordert wäre
- Unbekannte Technologien, welche das präzise Schätzen der Aufwände erschwerten
- Bug im XAML2CPP Tool

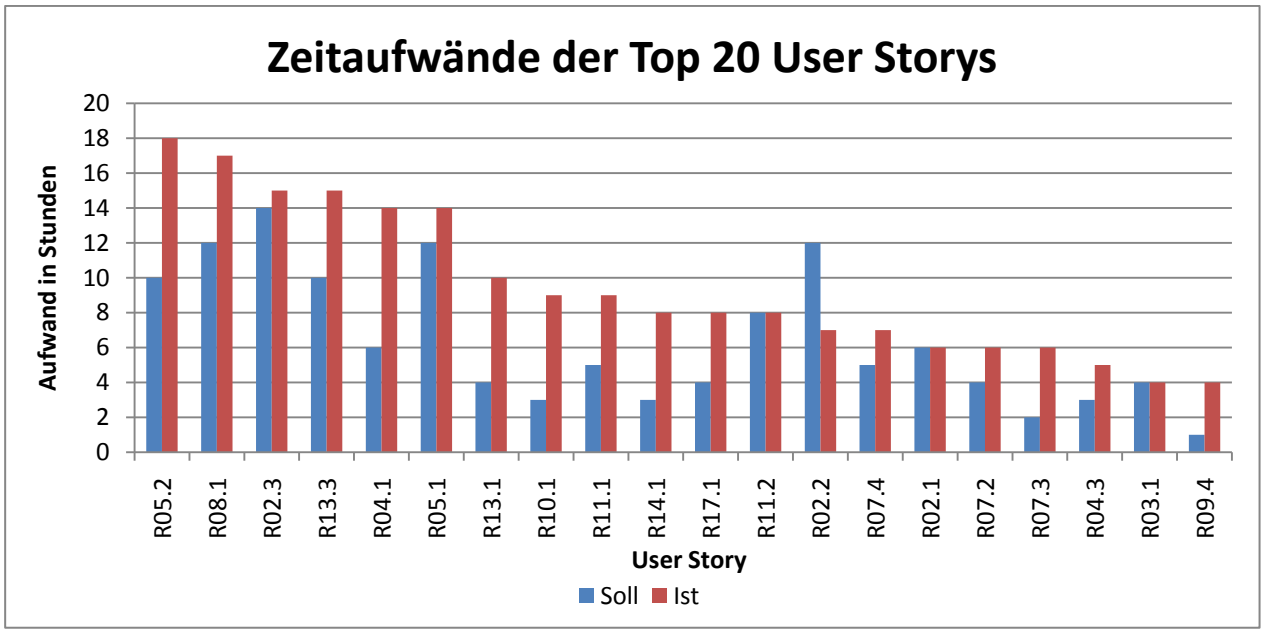


Abbildung 7: Zeitaufwände der Top20 User Storys

Id	Bezeichnung
R05.2	Bewegungsablauf abspielen
R08.1	Roboternetz aufbauen
R02.3	Integration in Visual Studio
R13.3	Bewegungsabläufe starten
R04.1	Steuern des Roboters
R05.1	Bewegungsablauf aufzeichnen
R13.1	Mit Roboter Controller verbinden
R10.1	Fehlerzustand anzeigen
R11.1	Alarme konfigurieren
R14.1	Betriebsdaten anzeigen
R17.1	Notaus anzeigen
R11.2	Alarme behandeln
R02.2	Code Generierung anhand WSDL
R07.4	Auflösen des Pairing

R02.1	WSDL Generierung anhand Assembly
R07.2	Verbindungsaufbau zu Roboter Controller
R07.3	Automatische Wiederaufnahme der Verbindung
R04.3	Steuerung über Tastatur
R03.1	Integration des Code Generators in MSBuild/TeamBuild

Tabelle 12: Liste der Top 20 User Storys

Teil 2: Iterationen

9 Inception

9.1 Inception 1

9.1.1 Ziele

In der Inception Phase geht es hauptsächlich um die definitive Definition der Aufgabenstellung und das Einrichten der Arbeitsumgebung. Dazu mussten folgende Aufgaben erledigt werden:

- Kennenlernen und Besprechen der Anforderungen beim Kickoff Meeting mit Zühlke
- TFS 2010 Installation & Evaluation
- Fragenkatalog für Interview mit MKL zusammenstellen, welcher neben den Anforderungsdokumenten von Zühlke als Basis für die Software Requirements Specification dient.
- Technologiestudium: Kennenlernen der Windows Embedded Plattform
- Installation Entwicklungsumgebung

9.1.2 Ergebnisse

Anhand eines Testprojektes konnte mit dem TFS experimentiert werden und so die Tauglichkeit für unser Projekt geprüft werden. Dabei wurde besonders Wert auf die Verwaltung der Dokumente, Issue Tracking, Continuous Integration und das Task Management gelegt. Der TFS baut auf so genannten Work Items auf, welche z.B. User Storys, Tasks, Issues, Bugs oder Test Cases sind. Diese können beliebig miteinander verbunden oder in Hierarchien eingeordnet werden. Dies hat uns sehr gefallen und wir werden für dieses Projekt den TFS benutzen.

Die Installation der Entwicklungsumgebung stellte sich komplizierter heraus als angenommen. Diverse Softwarepakete, Servicepacks und Updates mussten in der richtigen Reihenfolge installiert werden, dass diese auch richtig funktionierten. Die Installation wurde im Dokument *\07_Technical Resources\Installationsanleitung.docx* festgehalten.

Beim Kickoff Meeting wurde uns die Richtung, in welche das Projekt geht klar. Trotzdem gab es einige Unklarheiten zum Endprodukt. Die Unklarheiten konnten mit einem vorbereiteten Interview (*\04_Requirements Analyse\Requirements Analyse.docx*) mit Mario Klessascheck geklärt werden. Mithilfe des Fragenkatalogs und den Aufgabenstellungen der HSR und Zühlke soll in der Elaboration 1 Phase die Software Requirements Specification erstellt werden.

9.1.3 Reflexion

Wir haben uns sehr auf das Kickoff Meeting gefreut, da wir trotz Aufgabenstellung die Aufgabe noch nicht wirklich verstanden haben. Das Kickoff Meeting schaffte viel Klarheit, wo zugleich ein Termin für ein Interview vereinbart wurde. Das Interview mit Mario Klessascheck war sehr interessant und klärte weitergehende Unklarheiten, so dass wir für die Anforderungsspezifikation bereit waren.

10 Elaboration

10.1 Elaboration 1

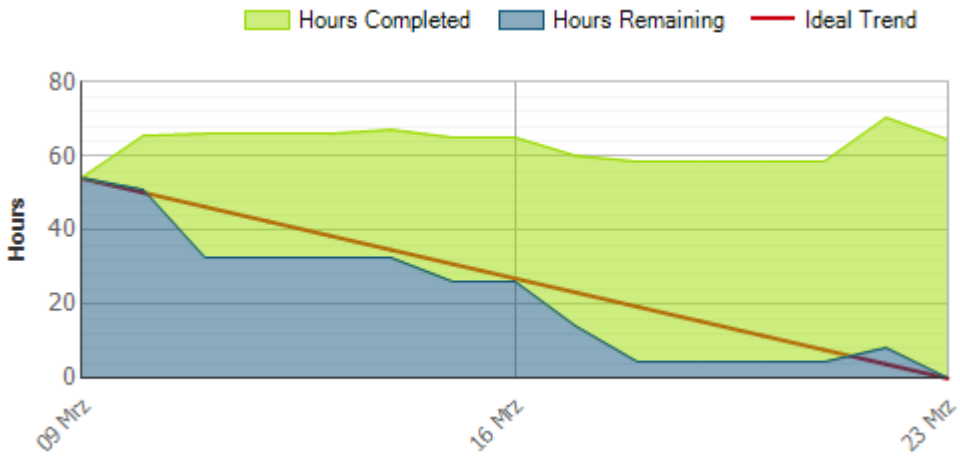


Abbildung 8: Burn Down Elaboration 1

10.1.1 Ziele

Das Hauptziel der Elaboration 1 Phase ist das Erstellen der Software Requirements Specification, welche von Zühlke bestätigt werden muss, so dass in der zweiten Elaboration Phase der Prototyp ausgearbeitet werden kann. Da nun die Technologien klar definiert sind, galt es zugleich weiteres Technologiestudium zu den spezifischen Themen zu machen. Des Weiteren wurde anhand der Aufgabenstellung eine Domainanalyse gemacht. Folgende Aufgaben galt es zu erledigen:

- Erstellen der Software Requirements Specification anhand des Interviews und bereits erhaltene Aufgabenstellungen
- Use Case Diagramm erstellen
- Analyse: Konzeptionelles Modell, State Diagram
- Technologiestudium

10.1.2 Ergebnisse

Anhand des Interviews und der beiden Anforderungsdokumente konnte das Software Requirements Document (*\04_Requirements Analyse\Requirements Analyse.docx*) und das Use Case Diagramm erstellt werden. Das Aufzeichnen des konzeptionellen Modells gab uns einen anderen Blickwinkel auf die Aufgabenstellung, worauf weitere Fragen auftauchten, welche gleichzeitig in die Software Requirements Specification floss. Bis auf kleine offene Punkte bezüglich der Roboter- bzw. Remote Controller Anwendung konnte die Anforderungsspezifikation abgeschlossen werden.

10.1.3 Reflexion

Microsoft stellt viele sehr gut dokumentierte Tutorials zur Verfügung. Dies ermöglichte das Erstellen und Deployen unseres ersten Windows Embedded OSDesign, welches bereits Silverlight for Embedded unterstützte. Das OS Konnte bereits mit einer vorgefertigten Silverlight Testapplikation getestet werden.

Besonders interessant war das Erstellen des konzeptionellen Modells. Wir haben die Problemstellung aus einem völlig anderen Winkel betrachtet und sind dabei auf weitere Unklarheiten gestossen, welche zugleich korrigiert werden konnten.

10.2 Elaboration 2

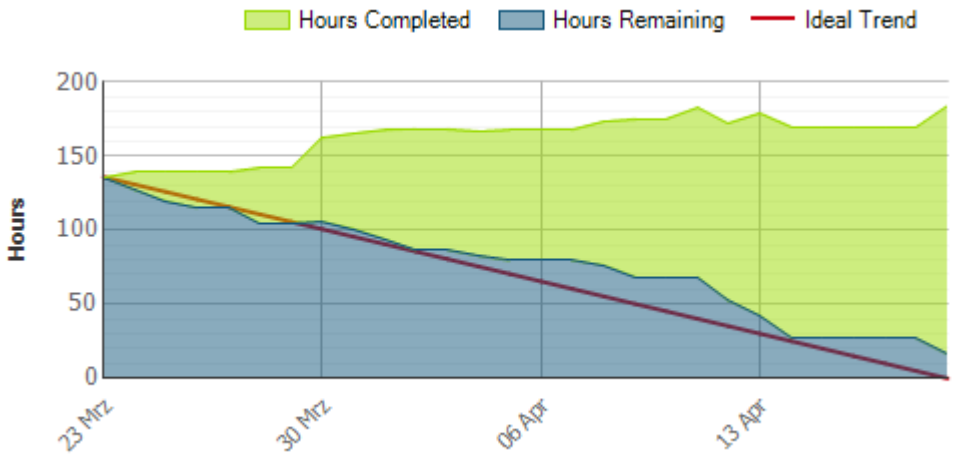


Abbildung 9: Burn Down Elaboration 2

10.2.1 Ziele

Das Ziel der Elaboration 2 ist die Implementation des Prototypen, bei dem ein Silverlight for Embedded UI und der DPWS-Stack implementiert wird. Dies sind die beiden wichtigsten Komponenten, welche am meisten Risikofaktoren besitzen. Dabei sollen diese Risiken mithilfe des Prototyps minimiert werden. Dazu gehören folgende Aufgaben:

- Implementation DPWS-Stack
- Prototyp: Implementation Silverlight UI
- Prototyp: Implementation eines Clients auf Windows Mobile
- Hardware Testumgebung einrichten
- Zwischenpräsentation bei Zühlke: Demonstration „Proof of concept“ mit Prototyp

10.2.2 Ergebnisse

Der DPWS-Stack wurde vom .NET Micro Framework portiert, worauf am Ende der Elaboration 2 Phase ein voll funktionsfähiger DPWS-Stack zur Verfügung stand. Beim Prototyp wurde der DPWS-Stack zum ersten Mal eingesetzt. Ein Silverlight UI auf dem Prototyp-Device visualisierte die empfangenen One-Way und Two-Way Anfragen. Des Weiteren konnte über das UI ein Event ausgelöst werden, welcher auf den registrierten Clients eine Callbackmethode aufruft.

Da bei der Entwicklung Probleme mit dem Emulator auftraten, wurde eine Hardware Infrastruktur mit einer eBox 3400 und einem WLAN-Router aufgebaut, welche zugleich als Testumgebung verwendet werden kann.

Mit dem Prototyp gelang der Architekturdurchstich, welcher alle technisch kritischen Risiken abdeckt. Nun kann die Construction Phase beginnen, wo die Features des Roboter Controller Showcase implementiert werden.

10.2.3 Reflexion

Während der Portierung mussten wir feststellen, dass man bei der Entwicklung mit Embedded Geräten ohne Hardware so gut wie keine Chance hat. Zum einen sind die Emulatoren extrem langsam (vergleichbar mit 200Mhz, die eBox hat 500Mhz). Zudem können Netzwerkkarten nicht zu 100% emuliert werden. So können zum Beispiel keine Multicast Pakete empfangen werden. Nach ca. 2 Tagen WS-Discovery Hello-Message im Debugger suchen haben wir in den Release Notes eines älteren Emulators den Vermerk gefunden, dass Multicastpakete nicht empfangen werden können. Daraufhin haben wir uns die benötigte Hardware besorgt, um damit eine Testumgebung einzurichten. Mit der Hardware konnten wir viel effizienter entwickeln, da keine Emulator-Probleme mehr auftraten. Trotzdem muss man extrem aufpassen, dass die Netzwerkeinstellungen überall korrekt konfiguriert sind. Die Einstellungen sind sehr versteckt und müssen Teils doppelt ge-

macht werden. Diverse Stolpersteine, Tipps und Tricks sind dabei im technischen Bericht im Kapitel „Technische Erkenntnisse“ dokumentiert.

11 Construction

Da in der Elaboration Phasen alle technisch kritische Risiken beseitigt wurden, werden in der Construction Phase die Funktionen des Showcase implementiert.

11.1 Construction 1

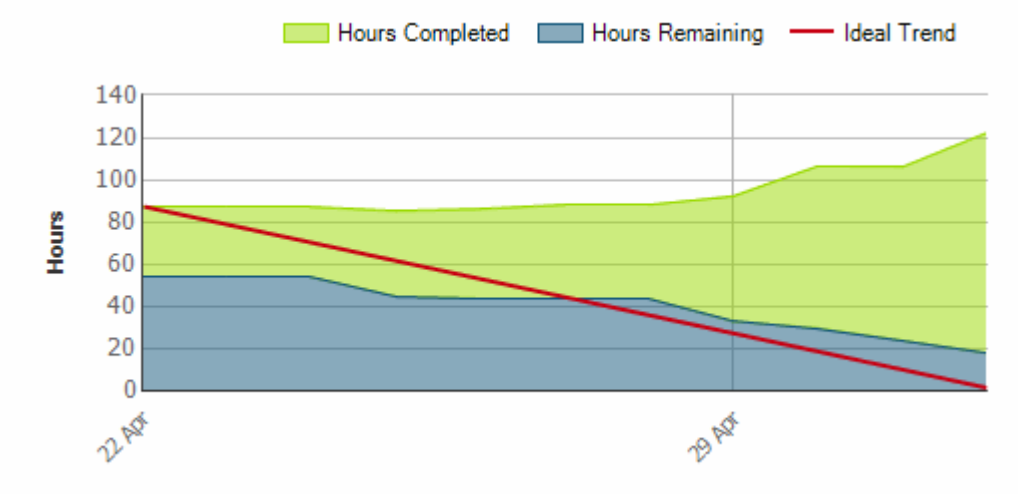


Abbildung 10: Burn Down Construction 1

11.1.1 Ziele

- Aufbau Softwarearchitektur für den Roboter Controller Showcase
- GUI Design Roboter- bzw. Remote Controller
- Dokumentieren: Software Architecure Document, Anpassungen an Projektplan
- Web Service Definition
- Behebung Code-Analysis Warnungen aus DPWS-Stack

11.1.2 Ergebnisse

Beide GUI Designs konnten abgeschlossen werden. Auf dem Roboter Controller konnten einige schöne Animationseffekte eingebaut werden.

Des Weiteren wurde der Projektplan überarbeitet wobei die geplanten Funktionen entsprechend auf die Construction Phasen verteilt wurden.

Im Pair Programming wurden die Grundrisse der Softwarearchitektur entwickelt und alle wichtigen Interfaces definiert, so dass ein paralleles Arbeiten möglich wurde. Da die Serviceschnittstelle sehr zentral ist, wurde diese bereits zu Beginn erstellt.

11.1.3 Reflexion

Das Entwerfen eines Designs für mobile Geräte ist besonders wegen der unterschiedlichen Displaygrössen und Auflösungen eine Herausforderung. Visual Studio bietet dafür so genannte Form Factors, jedoch existiert kein Form Factor für unser HTC HD2. Wir mussten die Erfahrung machen, dass man als erstes den Form Factor richtig konfigurieren muss, da ansonsten unter Umständen das ganze User Interface verzogen wird. Natürlich unterstützen die Emulatoren unsere Auflösung vom HTC HD2 nicht, so dass das User Interface im Debugger nie korrekt dargestellt wird. Dies ist etwas schade, doch man kann auf eine solche Funktion gut verzichten.

11.2 Construction 2

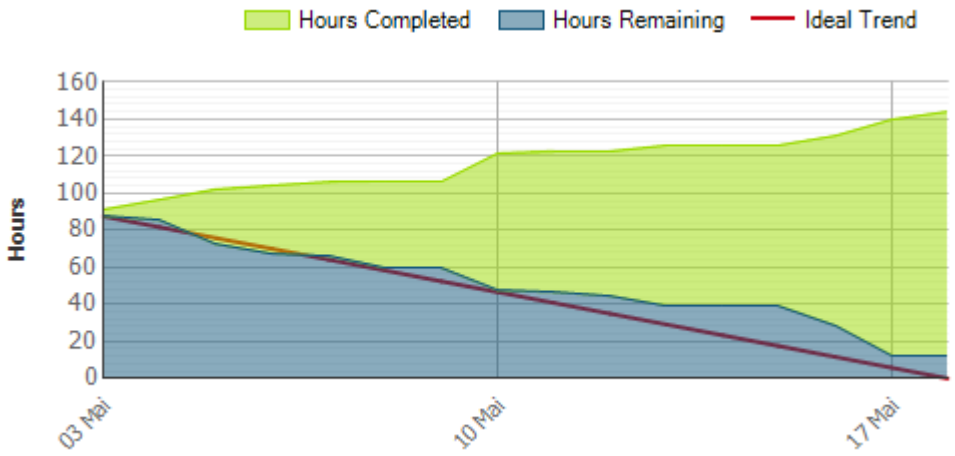


Abbildung 11: Burn Down Construction 2

11.2.1 User Storys

Folgende User Storys wurden in der Construction Phase 2 implementiert:

- R13.1: Mit Roboter Controller verbinden
- R13.2: Roboter steuern
- R14.1: Betriebsdaten anzeigen
- R15.1: Fehlerzustände anzeigen
- R04.1: Steuern des Roboters
- R06.1: Betriebsdaten führen
- R06.2: Betriebsdaten über Schnittstelle verfügbar machen
- R10.1: Fehlerzustand anzeigen
- R10.2: Fehlerzustände dem Remote Controller weiterleiten
- R07.2: Verbindungsaufbau zu Roboter Controller
- R07.3: Automatische Wiederaufnahme der Verbindung
- S01.1: Abrufen der Betriebsdaten
- S01.2: Bewegen der Motoren
- S01.3: Ein- bzw. ausschalten der LED
- R04.2: Steuerung über Touchinterface

11.2.2 Ergebnis

Der Roboter ist in diesem Zustand vollständig über DPWS oder direkt am Tochpanel steuerbar. Der Gerätetreiber und Roboter wurde früher als geplant von Zühlke geliefert und konnte innert einer Stunde problemlos integriert werden.

11.3 Construction 3

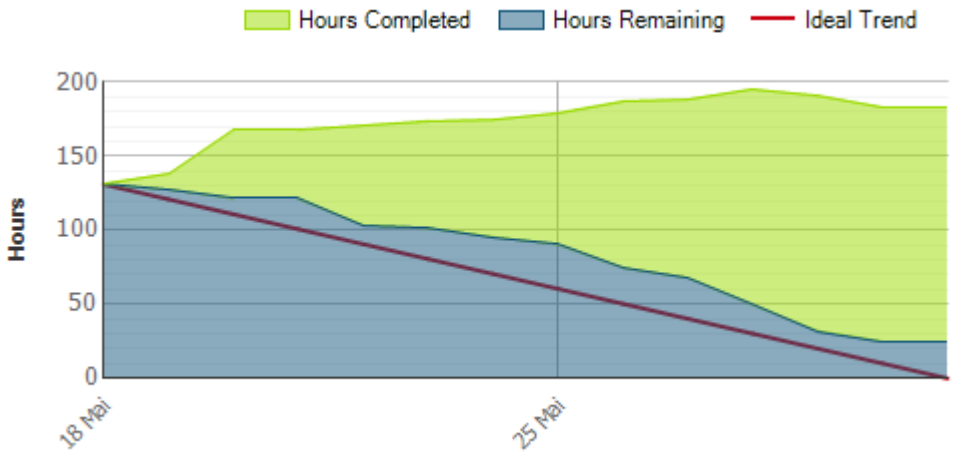


Abbildung 12: Burn Down Construction 3

In der Construction 3 Phase lag das Hauptaugenmerk auf der Implementation des Notaus, der Alarme und dem Aufnehmen und Abspielen der Bewegungsabläufe. In der Mitte der Construction 3 Phase soll der aktuelle Stand der Applikation bei Zühlke präsentiert werden.

11.3.1 User Storys

- S01.4: Notaus auslösen/aufheben
- R11.1: Alarme konfigurieren
- R11.2: Alarme behandeln
- R11.3: Alarme dem Remote Controller weiterleiten
- R07.1: Schutz nach Geräteklassen
- R04.3: Steuerung über Tastatur
- R08.1: Roboternetz aufbauen
- R08.2: Roboternetz verlassen
- R09.1: Notaus manuell auslösen
- R09.2: Grenze für automatisches Notaus konfigurieren
- R09.3: Automatischer Notaus auslösen
- R09.4: Notaus dem Remote Controller weiterleiten
- R09.5: Notaus dem ganzen Roboternetz weiterleiten
- R09.6: Notaus aufheben
- R16.1: Alarme anzeigen
- R17.1: Notaus anzeigen
- R17.2: Notaus manuell auslösen
- R17.3: Notaus auflösen
- R05.1: Bewegungsablauf aufzeichnen
- R05.2: Bewegungsablauf abspielen
- R05.3: Bewegungsablauf rückwärts abspielen
- R05.4: Bewegungsablauf löschen

11.3.2 Ergebnisse

Ein Notaus stoppt die aktuelle Bewegung und blockiert alle Eingaben bis das Notaus quittiert wird. Dabei wird das Notaus an andere Roboter im LAN gemeldet, so dass diese auch blockiert werden. Von einem beliebigen Roboter Controller kann das Notaus aufgehoben werden, was auch wieder den anderen Geräten im Netzwerk weitergeleitet wird.

Des Weiteren kann nun der Roboter über die Tastatur gesteuert werden. Die Konfiguration kann in der Applikationskonfigurationsdatei umkonfiguriert werden.

Bewegungsabläufe lassen sich auf dem Roboter Controller aufnehmen und werden in einer XML-Datei abgespeichert. Das XML wurde möglichst leserfreundlich gestaltet, so dass Bewegungsabläufe auch über XML definiert werden können.

Die Präsentation bei Zühlke konnte erfolgreich durchgeführt werden. Mario Klessascheck und Michael Koster waren sehr zufrieden mit der Arbeit.

11.3.3 Reflexion

Bis anhin wurden lediglich Strings zwischen C# und C++ ausgetauscht. Für die Bewegungsabläufe mussten jedoch „echte" Objekte übermittelt werden. Dies ist kein einfaches Unterfangen und kostete viel Zeit und Geduld. Leider hat die Anwendung immer noch ein Memoryleak, welches bis jetzt noch nicht behoben werden konnte. Solange nicht mehr als sieben Bewegungsabläufe gespeichert sind, funktioniert es allerdings einwandfrei.

11.4 Construction 4

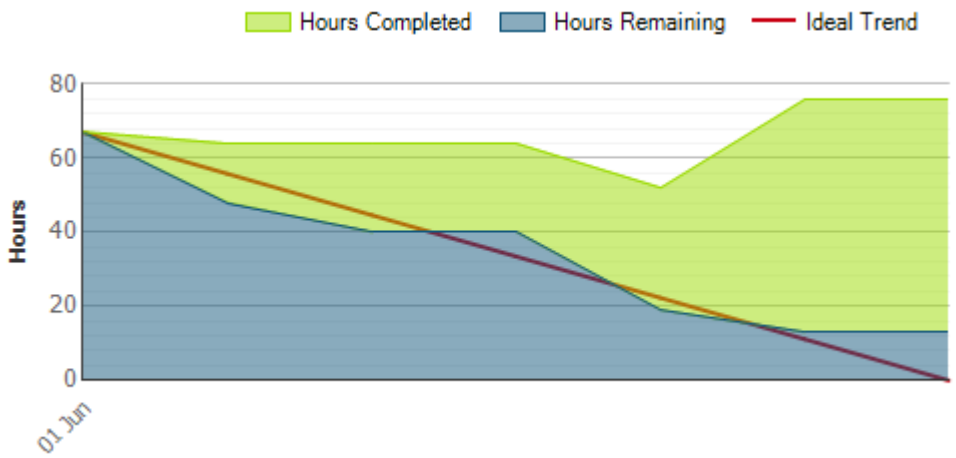


Abbildung 13: Burn Down Construction 4

In der Construction Phase 4 wurden nur noch kleine Funktionalitäten von den beiden Showcaseapplikationen implementiert. Es wird auf die Visual Studio und MSBuild Integration fokusiert.

11.4.1 User Storys

- R13.3: Bewegungsabläufe starten
- R03.1: Integration des Code Generators in MSBuild/TeamBuild
- R07.4: Auflösen des Pairing
- R12.1: Fehlerzustände loggen
- R12.2: Alarme loggen
- R12.3: Notaus loggen
- R02.1: WSDL Generierung anhand Assembly
- R02.2: Code Generierung anhand WSDL
- R02.3: Integration in Visual Studio

11.4.2 Ergebnisse

Analog zum Micro Framework (MFSvcUtil.exe) gibt es nun für den DPWS-Stack Dpws.SvcUtil.exe, mit dem diverse DPWS-Klassen generiert werden können. Die Funktionalität des Kommandozeilentools wurde zugleich ins Visual Studio integriert, was nun ein komfortables Generieren der Serviceproxys und Stubs ermöglicht.

11.4.3 Reflexion

Das Portieren des MFSvcUtil vom Micro Framework bereitete keine grossen Probleme. Die gemachten Änderungen wurden im technischen Bericht festgehalten. Das Entwickeln

des Visual Studio Add-ins war jedoch umständlicher als erwartet. Gemachte Erfahrungen wurden wiederum im technischen Bericht unter „Technische Erkenntnisse“ dokumentiert.

12 Transition

12.1 Ziele

- Abstract fertigstellen
- Management Summary fertigstellen
- Poster fertigstellen
- Dokumentation fertigstellen und ausdrucken
- Persönliche Erfahrungsberichte schreiben
- DVD für Abgabe erstellen
- Source Code wo nötig fertig dokumentieren
- Setup Visual Studio Add-in und Remote Controller Applikation erstellen

12.2 Ergebnisse

Wie geplant konnten alle Dokumente und Abgabeartefakte pünktlich fertiggestellt und abgegeben werden.

12.3 Reflexion

Da der geplante Code-Freeze tatsächlich eingehalten wurde, konnten wir uns in der Transition Phase zu hundert Prozent den Abgabearbeiten widmen und mussten keine Features mehr implementieren.

Teil 3: Erfahrungsberichte

13 Erfahrungsberichte

13.1 Bericht Stefan Züger

Die 16 Arbeitswochen unserer Bachelorarbeit sind bereits vorüber, weshalb ich die Gelegenheit nutzen möchte meine Erkenntnisse und Erfahrungen kurz festzuhalten.

Als wir die erste Aufgabenstellung von Zühlke erhielten, war für mich Vieles unklar und ich konnte mir noch nicht genau vorstellen, wie das Resultat unserer Arbeit aussehen soll. Viele der einzusetzenden Technologien waren für mich neu. Umso mehr war ich auf das Kickoff Meeting bei Zühlke in Schlieren gespannt. Nach dem ersten Meeting und einem anschliessenden Interview mit Mario Klessascheck konnten dann aber alle Unklarheiten beseitigt werden und ich freute mich auf die bevorstehende interessante Zeit.

In einer ersten Phase galt es alle neue Technologien wie zum Beispiel Windows Mobile, Windows Embedded oder Silverlight for Embedded genauer kennen zu lernen. Gerade bei Silverlight for Embedded gab es anfangs einige Schwierigkeiten, da dieses Thema sehr neu und noch nicht sehr verbreitet ist. Trotzdem konnten wir immer alle Probleme lösen, nicht zuletzt dank der kompetenten Hilfestellungen von Michael Koster.

Die Projektplanung nach RUP mit einem umfangreichen Architekturprototyp über alle Layer hat sich als sehr erfolgreich herausgestellt. So konnten wir zur Hälfte des Projekts einen Prototyp präsentieren, welcher alle technischen Risiken abdeckte. Mit der Gewissheit im Rücken, dass alle technischen Risiken geklärt sind, konnten wir uns in der Construction Phase voll auf die Umsetzung der Features konzentrieren. Der Einsatz des Team Foundation Server 2010 lohnte sich in allen Belangen. Anfangs bedeutete dies zwar einen kleinen Mehraufwand bis alles eingerichtet war, doch war es während dem Projekt doch sehr hilfreich alle User Storys, Tasks, Arbeitszeiten, Bugs, Issues etc. damit zu verwalten. Bei einem weiteren Projekt dieser Art würde ich den TFS bestimmt wieder einsetzen.

Rückblickend kann ich für mich sagen, dass mich die Bachelorarbeit in allen Belangen voll zufrieden gestellt hat. Die Arbeit war fordernd und anspruchsvoll, doch stets äusserst interessant. Ich habe während diesen 16 Wochen sehr viele neue Technologien und Konzepte kennengelernt, welche ich bestimmt für meine zukünftige Tätigkeit als Software Ingineur gebrauchen kann.

Für die hervorragende Zusammenarbeit mit Mario Klessascheck, Michael Koster und unseren Betreuern Hansjörg Huser und Jürg Jucker möchte ich mich herzlich bei allen Beteiligten bedanken. Sie war stets konstruktiv und weiter motivierend. Des Weiteren möchte ich meinem Bachelorarbeitspartner Tobias Zürcher für die ausgezeichnete Zusammenarbeit, die konstruktiven Diskussionen und seine geleistete Arbeit danken. Es freut mich sehr, dass wir beide nicht zuletzt dank der Bachelorarbeit einen Job bei Zühlke als Software Ingineure erhalten haben und somit auch in Zukunft zusammenarbeiten können.

13.2 Bericht Tobias Zürcher

16 Wochen Bachelorarbeit sind nun vorbei. Gerne blicke ich auf diese intensive und interessante Zeit zurück.

Zu Beginn wusste ich überhaupt nicht, auf was ich mich einlasse, da die Arbeit beim Bewerbungsprozess sehr unpräzise formuliert war. Selbst nach dem Kickoff Meeting gab es noch viele Unklarheiten, welche wir selber klären mussten und in einem Software Specification Document festhielten. Dieses sehr realitätsgetreue Szenario hat mich sehr motiviert zu arbeiten, ich denke dass dies auch eine gute Übung für den Einstieg in das Berufsleben ist. Bis anhin war das Erstellen einer Softwarespezifikation für mich persönlich eher eine Qual, da alles zu theoretisch war. Bei der Bachelorarbeit ging es nun wirklich um ein Endprodukt, welches produktiv verwendet werden soll. Beim Interview gab uns Mario Klessascheck weitere Tipps und gab uns auch einen interessanten Einblick in die Zühlke, wie dort Vorgegangen wird.

Speziell an diesem Projekt war, dass ich sozusagen nur mit neuen Technologien gearbeitet habe. Dies erschwerte nicht nur die Implementierung, sondern begann bereits bei der Projektplanung, da Arbeitspakete mit fast keinen Anhaltspunkten/Erfahrungen geschätzt werden mussten. Besonders die Implementation mit Silverlight for Embedded kostete viel Energie und Zeit. Die Freude war jedoch umso grösser, als es endlich funktionierte, und ich wollte umso mehr UI Spielereien umsetzen.

Zu Beginn haben wir etwas mehr Zeit in die Installation und Konfiguration des TFS 2010 investiert. Dies hat sich enorm gelohnt, da viele Werkzeuge uns die Arbeit erleichterten. Anforderungen sowie Tasks, Issues und Bugs wurden im TFS erfasst und konnten im Excel wiederverwendet werden. Dies erleichterte das Projektmanagement ungemein.

Ich bin mit der abwechslungsreichen und stets fordernden Bachelorarbeit sehr zufrieden. Bedanken möchte ich mich für die vorbildliche Betreuung von Prof. Hansjürg Huuser und seiner Stellvertretung Jürg Jucker. Auch bei Mario Klessascheck und Michael Koster von Zühlke Engineering AG möchte ich mich für die hervorragende Arbeit bedanken. Einen speziellen Dank geht mein Arbeitspartner Stefan Züger für die ausgezeichnete Zusammenarbeit. Es war bereits das vierte Projekt an der HSR welches ich mit ihm durchgeführt habe. Ich freue mich sehr, nicht zuletzt auch wegen der Bachelorarbeit, am 20. September mit Ihm meine neue Arbeitsstelle bei Zühlke zu beginnen.

Teil 4: Sitzungsprotokolle

14 Sitzungsprotokolle

14.1 Sitzungsprotokoll vom 24.02.2010

Allgemeine Informationen

Betreff	Kickoff Meeting
Ort	Zühlke Engineering, Schlieren
Datum	Mittwoch, 24.02.2010, 09:00 – 11:00

Teilnehmer

Name	Rolle	Kürzel
Dipl. Ing. Mario Klessascheck	Auftraggeber	MKL
Dipl. Ing. Michael Koster	Technischer Experte	MKO
Prof. Hansjörg Huser	Betreuer	HH
Dipl. Ing. Jürg Jucker	Betreuer	JJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Kennenlernen der Beteiligten

erledigt

Bereits besprochene Anforderungen

Aufgabenstellung wurde nochmals erläutert.

Genaue Anforderungen sind noch nicht vorhanden. Folgende Priorisierung der Komponenten wurde aber bereits festgelegt:

1. Webserver & HMI mittels Silverlight for Embedded
   Technologien: Windows Embedded CE 6.0 R3, WCF, WSDL, SOAP 1.2, DPWS
2. Remote Controller
   Technologien: Windows Mobile 6.5 mit WinForms, evtl. Windows Mobile 7 mit Silverlight, Compact Framework, WCF

MKO wird SZ & TZ einen Terminvorschlag für das Interview machen.

Festlegen der nächsten Arbeitsschritte

Siehe Kapitel weiteres Vorgehen

Weiteres Vorgehen

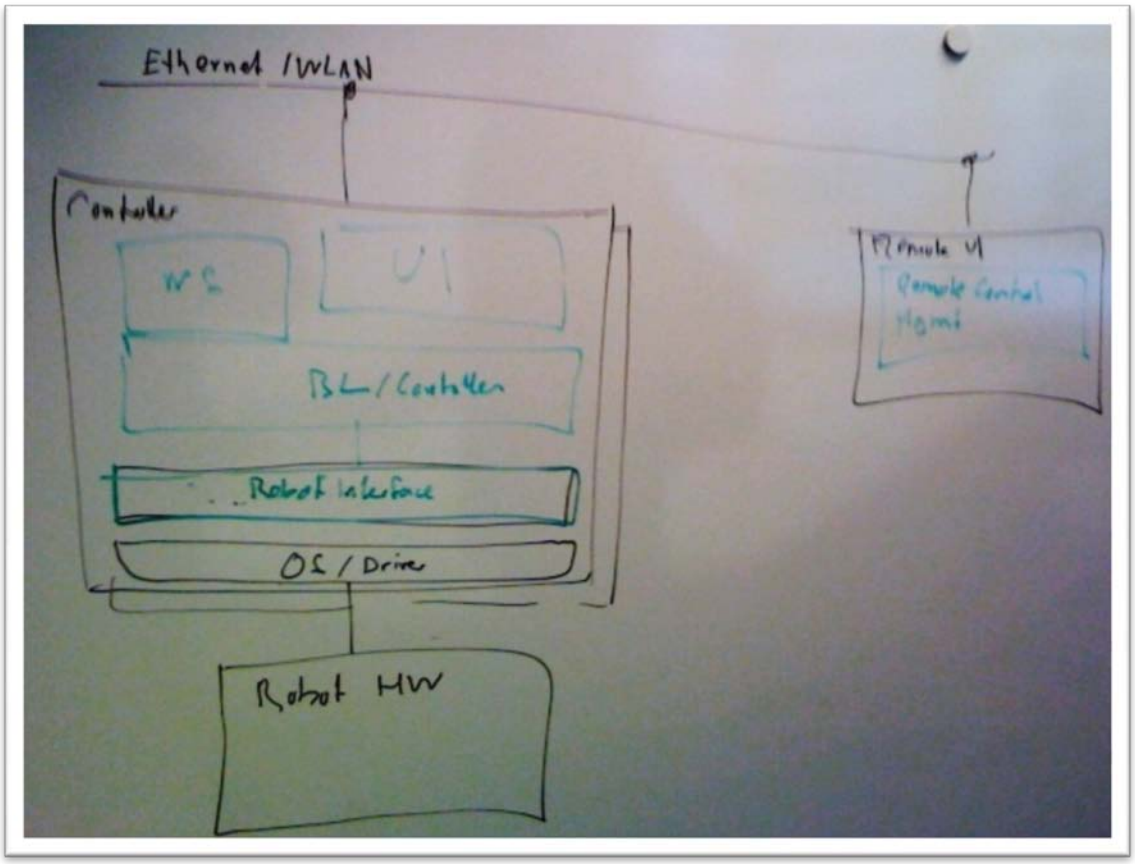
Task	Verantwortlich	Erledigt bis
Erste Version der Anforderungen	MKL	02.03.2010
Termin für Requirements Meeting mit Mario Klessascheck	SZ, TZ, MKL	02.03.2010
Linksammlung der wichtigsten Links zu Windows Embedded	MKO	26.03.2010

Nächster Termin

Datum: Donnerstag, 25.02.2010
Zeit & Ort: 09:00 – 10:00, HSR Raum 6.010

Anhang

Skizze der groben Systemarchitektur, gezeichnet von Michael Koster:



14.2 Sitzungsprotokoll vom 25.02.2010

Allgemeine Informationen

Betreff	Meeting #1
Ort	HSR, 6.010
Datum	Donnerstag, 25.02.2010, 09:00 -10:00

Teilnehmer

Name	Rolle	Kürzel
Prof. Hansjörg Huser	Betreuer	HH
Dipl. Ing. Jürg Jucker	Betreuer	JJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Besprechung der Vorgehensart

Vorgehen in zwei Stufen:

- **Stufe 1:** Requirements aufnehmen → Frageliste für Interview mit Mario erstellen
  - Funktionale Anforderungen
  - Nicht funktionale (WSDL-Standard, Security)
- **Stufe 2:** Technische Abklärungen (Fokus Web Service)
  - Managed Webserver → bereits vorhandene Implementationen studieren
  - Entwicklungsumgebung einrichten (Windows Embedded Platform, Compile-, Debug- & Deploy-Tools, Embedded Emulator für Debugging)

Bereits besprochene Anforderungen

Es soll ein Fragebogen für das Interview von Mario Klessascheck zur Definition der Anforderungen zusammengestellt werden. Folgende Anforderungen müssen im Interview geklärt werden.

- Service-Schnittstelle
  - Funktionale Anforderungen
  - Nicht funktionale Anforderungen → WSDL Standard/Protokollversion, Security
  - Technische Anforderungen
  - Reliability?
- Services
  - Commands → Befehle wie z.B. „Beweg Achse X“
  - Lesen von Daten des Roboters
  - Fehler/Aalarmzustand verwalten
- Funktionalität Controller
  - Zustandsverwaltung?

Projektplan

- Bis zur nächsten Sitzung soll ein grober Projektplan erstellt werden.
- In der Mitte des Projektes soll ein einfacher Prototyp erstellt sein, mit dem der Roboter rudimentär lokal oder per Handy gesteuert werden kann → technische Absicherung

Technologieabklärungen

- Compact-Framework verfügt nur über ein Subset der Funktionalitäten für WCF
- Windows Embedded emulierbar für Entwicklung? Wenn nicht → JJ hätte Gerät zur Verfügung
- Cassini & C# Webserver von Codeplex anschauen

Allgemeine Abklärungen bezüglich BA

- TFS wird als SCM verwendet (nicht wie in Aufgabenstellung beschriebene SVN)
- Projektmanagement muss nicht strikt nach RUP geführt werden. Studenten werden einige Aspekte aus RUP und Scrum kombinieren (optimiert auf dieses Projekt)
- Risiko-Analyse: nur Projektspezifische Risiken beschreiben

Windows Mobile Gerät

Zur Entwicklung des Remote Controllers wurde den Studenten ein HTC HD2 Mobiltelefon überreicht.

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Fragenkatalog erstellen für Interview	TZ, SZ	03.02.2010
Windows Embedded ISO von MSDN	JJ	25.02.2010
TFS-Logindaten für JJ & HH	TZ	04.03.2010
Cassini & C# Weberver Implementation studieren	TZ, SZ	04.03.2010

Nächster Termin

Datum: Donnerstag, 04.03.2010
Zeit & Ort: 17:00, HSR Raum 6.010

14.3 Sitzungsprotokoll vom 04.03.2010

Allgemeine Informationen

Betreff	Weekly Betreuer Meeting
Ort	6.010
Datum	Donnerstag, 04.03.2010, 17:00 – 18:30

Teilnehmer

Name	Rolle	Kürzel
Prof. Hansjörg Huser	Betreuer	HH
Dipl. Ing. Jürg Jucker	Betreuer	HJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

DPWS

Mögliche Varianten:

- Casini abändern
- Wrapper von nativem Code (→ prüfen ob dieser überhaupt auf Embedded läuft)
- Test-Host von WCF
- Microframework Implementierung portieren (Sourcecode in Micro-FW Porting Kit)

→ In Woche 5 diese Varianten untersuchen und einen Entscheid fällen.

- Es wird kein Webserver benötigt, lediglich Web Service.
- MFSvcUtil Source vorhanden
- Sourcecode ist unter Apache Lizenz → überprüfen was das heisst

Weitere Ideen als Anforderung von JJ & HJ

- Zustand per Software verwalten, da keine Sensoren vorhanden sind. Frage: Kann Strom gemessen werden?
- Problem: Initial-Zustand! Vorhanden?
- Alarm:
  - Definieren von min/max
  - Roboter lernen → Ø um Prozentwert oder x über- bzw. unterschritten
- Teaching → Befehlssequenz merken
  - Wie intelligent muss Teaching sein?
- Logging
  - Log von Zeit & Zustand z.B. alle 1s
  - Alarm-Log → Liste aller Alarme anzeigen
    - Abklären, was alles genau geloggt werden soll
    - Flapping verhindern

Nächste Schritte

HJ schlägt folgende Schritte vor:

1. Konzeptionelles Modell (Domain Model)
2. Anforderungsspezifikation
3. Service Interface

- Erster Prototyp braucht kein Discovery
- Evtl. Desktopclient zum Testen
- Konzeptionelles Modell von Roboter (Achsen etc.) und seinen Zuständen (State Diagramm)

Abklärungen

- Bis MS3 Treiber da?
- Ab wann brauchen wir Infrastruktur? → Risiko

Abwesenheiten

HJ ist für 3 Wochen weg, ab Woche 6 wieder da.

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Software Requirements Specification schreiben	TZ, SZ	14.03.2010
Gerätetreiber von Zühlke	MKL	19.05.2010

Nächster Termin

Datum: Donnerstag, 11.03.2010
Zeit & Ort: 14:00, HSR Raum 6.010

14.4 Sitzungsprotokoll vom 05.03.2010

Allgemeine Informationen

Betreff	Requirements Analyse Interview
Ort	HSR, Mensa
Datum	Freitag, 05.03.2010, 08:30 -10:30

Teilnehmer

Name	Rolle	Kürzel
Dipl. Ing. Mario Klessascheck	Auftraggeber	MKL
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Klären aller Fragen im Fragenkatalog
Das Meeting war ausschliesslich für die Beantwortung aller Fragen im Fragenkatalog reserviert. Dieses Ziel wurde erfüllt. Alle Antworten wurden direkt im bestehenden Fragenkatalog ergänzt: Siehe `03_Analyse\Fragenkatalog_Requirements_Analyse.docx`

Treiber von Zühlke
Für die Anbindung des Roboters am Windows Embedded System entwickelt Zühlke einen entsprechenden Gerätetreiber. Gemäss MKL können wir nicht vor April mit dem Treiber rechnen. Aus diesem Grund wurde entschieden für den 1. Prototypen auf dem Roboter Controller eine einfache Simulation des Roboters umzusetzen, der lediglich in der Konsole etwas ausgibt. Vorsichtshalber wurde deshalb die Integration des Roboters auf den MS5 vom 30.05.2010 gesetzt. Damit dieses Ziel erreicht werden kann, muss Zühlke den Treiber bis spätestens 19.05.2010 liefern.

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Software Requirements Specification schreiben	TZ, SZ	14.03.2010
Gerätetreiber von Zühlke	MKL	19.05.2010

Nächster Termin

Datum: Donnerstag, 11.03.2010
Zeit & Ort: 14:00, HSR Raum 6.010

14.5 Sitzungsprotokoll vom 11.03.2010

Allgemeine Informationen

Betreff	Weekly Betreuer Meeting
Ort	HSR, 6.010
Datum	Donnerstag, 11.03.2010, 14:00 – 15:00

Teilnehmer

Name	Rolle	Kürzel
Dipl. Ing. Jürg Jucker	Betreuer	JJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Projektplan

- Kurzes Review der Risiken im Projektplan. → alles i.O.
- Prototyp soll Notaus Funktionalität bereits enthalten
- Projektplan an JJ senden für Review

Anforderungsspezifikation

- Gemäss Absprache wird am Anfang der Software Requirements Specification ein Use Case Diagramm eingefügt.
- Requirement enthält neben Titel und Beschreibung eine Id, Priorität und Modalität
- Interface zum Treiber muss zwar festgelegt werden, ist aber zu technisch für eine SRS. Muss aber bis Ende Elaboration 2 definiert sein.

Weiteres

TFS Login mit JJ probiert → alles i.O.

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Projektplan an HH und JJ senden	SZ	11.03.2010
Anforderungsspezifikation am Mario Klessascheck senden	SZ / TZ	15.03.2010

Nächster Termin

Datum: Donnerstag, 18.03.2010
Zeit & Ort: 15:00, HSR Raum 6.010

14.6 Sitzungsprotokoll vom 18.03.2010

Allgemeine Informationen

Betreff	Weekly Betreuer Meeting
Ort	HSR, 6.010
Datum	Donnerstag, 18.03.2010, 15:00 – 16:00

Teilnehmer

Name	Rolle	Kürzel
Dipl. Ing. Jürg Jucker	Betreuer	JJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Projektplan

- i.O, offene Punkte werden erst in späteren Iterationen vervollständigt

Anforderungsspezifikation

- Anforderungsspezifikation an Zühlke geschickt, warten auf Feedback
- JJ ist mit Anforderungsspezifikation zufrieden

Domainanalyse

- System Sequenzdiagramme sind zu trivial → können weggelassen werden
  - Kommunikation zwischen den Subsytemen wird später im SAD mit Sequenzdiagrammen visualisiert
- Activity Diagramm ungeeignet, da Prozess trivial → kann auch weggelassen werden
- Domain Modell / Konzeptionelles Modell
  - Einzelne Attribute haben gefehlt
    - Roboter: SerieNr/Bezeichnung, IP
    - IP bei Remote- bzw. RoboterController als Identifikation
    - Ausnahme: Timestamp & Textfeld
    - Alarm: Schwellen
    - Notaus: Text
- State Diagram
  - Mit ein zwei Sätzen State Diagram beschreiben
    - Alarmüberprüfung: Wann? Motor bei State „BewegeMotor“
    - Perspektive: aus Sicht des Roboters

Treiberschnittstelle

Mit Zühlke abklären, wie die Treiberschnittstelle aussieht → Risiko vermindern

Weiteres Vorgehen

Warten auf Feedback von Zühlke. In Zwischenzeit kann bereits die Implementierung respektive Analyse vom DPWS-Stack in Angriff genommen werden (→ für Architekturprototyp MS3, 18.04.2010).

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Treiberschnittstelle Details soweit wie möglich abklären	TZ	09.05.2010
Korrekturen der Domainanalyse vornehmen	SZ	09.05.2010

Nächster Termin

Datum: Donnerstag, 01.04.2010
Zeit & Ort: 15:00, HSR Raum 6.010

14.7 Sitzungsprotokoll vom 01.04.2010

Allgemeine Informationen

Betreff	Weekly Betreuer Meeting
Ort	HSR, 6.010
Datum	Donnerstag, 01.04.2010, 15:00 – 16:00

Teilnehmer

Name	Rolle	Kürzel
Prof. Hansjörg Huser	Betreuer	HH
Dipl. Ing. Jürg Jucker	Betreuer	JJ
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Projektplan/Risiken besprechen

- Projektplan und Risiko Management wurde nochmals mit HH besprochen. → alles i.O.

Anforderungsspezifikation

- HH ist mit Anforderungsspezifikation zufrieden
- Zweites Feedback von Zühlke immer noch ausstehend

Domainanalyse

- Domain Modell nochmals mit HH besprochen
  - HH ist sich nicht sicher, ob die dynamischen Beziehungen zwischen den konzeptionellen Klassen gezeichnet werden sollen. Des Weiteren hat er auch vorgeschlagen nur diejenigen Klassen zu modellieren, welche für das Software System von Interesse sind. Dies muss aber nochmals genauer in der Literatur nachgeschlagen werden.

Aktueller Stand

- Aktueller Stand an HH erläutert. Aktuell gibt es noch ein wenig Probleme mit SL4E. Dazu hat aber Michael Koster bereits Lösungsvorschläge geschickt, welche es nun gilt umzusetzen.
- Prototyp Entwicklung hat im Moment höchste Priorität

Weiteres Vorgehen

Task	Verantwortlich	Erledigt bis
Domain Model Theorie im Larman Buch nachlesen	TZ / SZ	08.04.2010

Nächster Termin

Datum: Donnerstag, 01.04.2010
Zeit & Ort: 15:00, HSR Raum 6.010

14.8 Sitzungsprotokoll vom 22.04.2010

Allgemeine Informationen

Betreff	Demonstration Prototyp
Ort	Zühlke Engineering, Schlieren
Datum	Donnerstag, 22.04.2010, 09:00 – 12:00

Teilnehmer

Name	Rolle	Kürzel
Dipl. Ing. Michael Koster	Technischer Experte	MKO
Stefan Züger	Student	SZ
Tobias Zürcher	Student	TZ

Traktanden

Präsentation zur Entwicklung des Prototyps

- DPWS Implementierung
  - Entscheid zum Portieren des MF Codes ist für MKO i.O.
  - Einsatz von Log4Net i.O. für MKO. Er hat aber vorgeschlagen, das Logging zu kapseln, damit der Code nicht auf eine bestimmte Logging Lösung eingeschränkt wird. → last prio
  - Proxy Generierung egal von welcher Source (Assembly oder WSDL) → HSR intern abklären welcher Ansatz geeigneter ist
- Silverlight for Embedded
  - Einsatz von XAML2CPP i.O. für MKO
  - Beim MVVM mit nativer View darauf achten, dass das ViewModel unabhängig wird von der View
- Probleme/Fragen
  - Bei Inkonsistenz zwischen WS-Discovery und DPWS Spezifikation an WS-Discovery Spezifikation halten
  - Bei C++ Strings kann gemäss MKO der BSTR verwendet werden, wenn Silverlight for Embedded diesen intern auch braucht

Demonstration Prototyp

- MKO sehr zufrieden mit dem Resultat

Teil 5: Anhang

15 Verzeichnisse

15.1 Tabellenverzeichnis

15.2 Abbildungsverzeichnis

Test

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

1 Einleitung

Dieses Test Dokument beschreibt das Vorgehen beim Testen und die dabei durchgeführten Testfälle.

1.1 Testvorgehen

Zum Testen der Gesamtapplikation wurden die Testfälle für die beiden Komponenten Roboter Controller und Remote Controller separat definiert. Jeder Testfall ist so geschrieben, dass er immer mindestens eine funktionale Anforderung abdeckt. Die funktionalen Anforderungen an den DPWS-Stack wurden direkt mit den Testfällen zum Remote Controller abgedeckt.

1.2 Testumgebung

Als Testumgebung dient die an der HSR aufgebaute Hardware-Umgebung, welche folgendes umfasst:

- Roboter Controller installiert auf Toradex Embedded Board mit Windows Embedded CE 6.0 R3 und angeschlossenem Touchpanel
- Playtastic NC-1425 Roboter angeschlossen am Toradex Board
- Zweite Roboter Controller Applikation installiert auf eBox-4300 mit Windows Embedded CE 6.0 R3 zum Testen der Roboternetz Features
- Remote Controller installiert auf HTC HD2 Mobilgerät mit Windows Mobile 6.5
- Eigenes WLAN zur Kommunikation zwischen Remote Controller und Roboter Controller

Ein Durchführen der Systemtests auf den Emulatoren ist leider nicht möglich, da die kompletten, auf UDP Multicast basierten WS-Discovery Features nicht im Emulator funktionieren.

Die Tests basieren auf den Standard Tastaturbelegungen.

2 Systemtests Roboter Controller

2.1 Testfälle

Id	Beschreibung	Erwartetes Resultat	Requirement	OK?
1.1	Roboter über Touchinterface steuern	Alle Achsen lassen sich über das Touchpanel in beide Richtungen bewegen. Die LED wird mit einem Klick auf den LED Button eingeschalten. Beim erneuten Klick auf denselben Button, wird die LED wieder ausgeschalten. Das Touchpanel zeigt immer an, ob die LED ein oder aus ist.	R4.1, R4.2	OK
1.2	Roboter über Tastatur steuern	Alle Achsen lassen sich über die Tastatur in beide Richtungen bewegen. Die LED wird mit einem Klick auf die Taste „L“ ein- bzw. ausgeschalten. Das Touchpanel zeigt immer an, ob die LED ein oder aus ist.	R4.1, R4.3	OK
1.3	Roboterbefehle über Tastatur und Touchpanel absenden	Wenn der Roboter über das Touchpanel gesteuert wird, so bewegt sich eine Achse jeweils solange, wie der Benutzer ununterbrochen auf eine Taste drückt. Bei der Steuerung über die Tastatur sind die Achsen jeweils solange in Bewegung, wie eine Taste nach unten gedrückt wird.	R4.1	OK
1.4	Roboter Controller agiert als Remote Controller	Wenn der Roboter keine Verbindung zu einem Roboter herstellen kann, erscheint auf dem Touchpanel eine Meldung, dass kein Roboter angeschlossen ist. Danach erhält der Benutzer die Möglichkeit nach weiteren Roboter Controllern im Netz zu suchen und als Remote Controller einen anderen Roboter Controller zu steuern	R4.4	NOK
1.5	Bewegungsablauf aufzeichnen	Nach einem Klick auf den Record Button zeigt die Applikation dem Benutzer an, dass er sich jetzt im Aufnahmemodus befindet. Ab diesem Zeitpunkt werden alle getätigten Bewegungen und Änderungen der LED aufgezeichnet. Es spielt keine Rolle ob eine Aktion über das Touchpanel oder die Tastatur ausgeführt wurde.	R5.1	OK
1.6	Bewegungsablauf speichern	Nachdem die Aufnahme gestoppt wurde, erscheint auf dem Display ein Dialog zur Eingabe eines Namens. Beim Klick auf OK wird der Bewegungsablauf im Dateisystem abgespeichert.	R5.1	OK
1.7	Bewegungsablauf abspielen	Es wird eine Liste mit allen aufgezeichneten Bewegungsabläufen dargestellt. Nach Auswahl eines Bewegungsablaufes und Klick auf „Play“ wird der Ablauf genau so abgespielt, wie er aufgezeichnet wurde.	R5.2	OK

1.8	Bewegungsablauf rückwärts abspielen	Es wird eine Liste mit allen aufgezeichneten Bewegungsabläufen dargestellt. Nach Auswahl eines Bewegungsablaufes und Klick auf „Reverse“ wird ein aufgezeichneter Bewegungsablauf in umgekehrter Reihenfolge abgespielt, wie er aufgezeichnet wurde. Der Roboter wird genau in den gleichen Zustand zurückgebracht, wie er vorher war.	R5.3	NOK
1.9	Bewegungsabläufe löschen	Nach Auswahl eines Bewegungsablaufes und Klick auf „Löschen“ muss der Ablauf aus der Liste verschwinden und vom Dateisystem gelöscht worden sein.	R5.4	OK
1.10	Mehrere Bewegungsabläufe hintereinander aufnehmen	Beim Aufzeichnen von beliebig vielen Bewegungsabläufen verhält sich alles gleich, wie wenn man nur einen Ablauf aufgezeichnet hat. Die Liste mit den Abläufen wird stets mit dem neu aufgezeichneten Ablauf gefüllt.	R5.1	NOK
1.11	Betriebsdaten führen	Bei jeder Aktion des Roboters wird die Dauer der Bewegung gestoppt und zu der bereits aufgelaufenen Zeit dazu addiert. Die Betriebsdaten werden permanent im Dateisystem gespeichert, so dass sie beim Neustart der Applikation wieder geladen werden können.	R6.1	OK
1.12	Betriebsdaten über Schnittstelle verfügbar machen	Der Roboter stellt die laufend aktuellen Betriebsdaten über einen DPWS Service zur Verfügung	R6.2	OK
1.13	Schutz nach Geräteklasse	Der Roboter Controller antwortet nur auf Discovery Anfragen von gültigen Remote Controllern	R7.1	NOK
1.14	Verbindungsaufbau zu Roboter Controller	Der Roboter Controller kann jeweils nur von einem Remote Controller gesteuert werden. Eine Steuerung wird nur zugelassen, wenn der Remote Controller zuvor eine Verbindung aufgebaut hat und über eine gültige Session verfügt.	R7.2	OK
1.15	Automatische Wiederaufnahme der Verbindung	Die Verbindung zwischen Remote Controller und Roboter Controller wird automatisch wieder aufgebaut wenn die vorher aufgebaute Verbindung nicht korrekt abgebaut wurde.	R7.3	OK
1.16	Auflösen des Pairing	Nach einem Klick auf den „Disconnect“ Button auf dem Touchpanel wird die Verbindung zum Remote Controller sofort beendet. Der Remote Controller hat keine Möglichkeit mehr den Roboter weiter zu steuern ohne erneut eine Verbindung aufzubauen.	R7.4	OK
1.17	Roboternetz aufbauen	Sobald sich ein neuer Roboter Controller in einem Netz anmeldet, schliesst er sich mit dem bestehenden Roboternetz zusammen.	R8.1	OK

1.18	Roboternetz verlassen	Sobald ein Roboter Controller heruntergefahren wird, meldet er sich korrekt vom Roboternetz ab.	R8.2	OK
1.19	Notaus auslösen	Wird ein Notaus automatisch, über Tastatur, Touchpanel, Remote Controller oder übers Roboternetz ausgelöst, so werden alle Roboteraktionen per sofort gestoppt. Es kann nichts mehr gesteuert werden, bis das Notaus aufgelöst wurde.	R9.1, R9.3	OK
1.20	Notaus weiterleiten	Wird ein Notaus ausgelöst, so wird dieses automatisch dem Roboternetz und dem verbundenen Remote Controller weitergeleitet.	R9.4, R9.5	OK
1.21	Automatischer Notaus	Wird eine Roboterachse ununterbrochen länger gesteuert als das konfigurierte Limit für das Auto-Notaus, so wird sofort ein Notaus ausgelöst.	R9.2, R9.3	OK
1.22	Notaus aufheben	Wenn ein Notaus aufgelöst wird, so wird der Notaus auch automatisch auf allen Robotern im Roboternetz und auf allen vorhandenen Remote Controller aufgelöst.	R9.6	OK
1.23	Fehlerzustände behandeln und anzeigen	Tritt auf dem Roboter Controller ein Fehler im Treiber auf, so zeigt er den Fehler korrekt auf dem Display an und leitet ihn dem verbundenen Remote Controller weiter.	R10.1, R10.2	OK
1.24	Alarme behandeln und anzeigen	Wird von einer Achse oder LED das konfigurierte Alarm Limit überschritten, so wird eine Alarmmeldung auf dem Display angezeigt und die Meldung dem verbundenen Remote Controller weitergeleitet. Das gesetzte Alarmlimit wird zurückgesetzt und der Alarm tritt nicht erneut auf.	R11.1, R11.2, R11.3	OK
1.25	Logging	Im Service Log der Roboter Controller sind alle aufgetretenen Fehlerzustände, Alarme und Notaus mit Datum und Uhrzeit ersichtlich.	R12.1, R12.2, R12.3	OK

Tabelle 1: Testfälle Roboter Controller

2.2 Nicht erfolgreiche Testfälle

Id	Begründung
2.4	Feature nicht implementiert
2.8	Das rückwärts Abspielen funktioniert ohne Probleme. Leider wird der Roboter aber praktisch nie in korrekten Anfangszustand geführt, da die Robotermotoren nicht in beide Richtungen gleich schnell drehen. Wird eine Achse vorwärts zwei Sekunden lang nach unten bewegt, so können die Motoren wegen der geringen Leistung in zwei Sekunden nicht die gleiche Strecke nach oben zurücklegen.
2.10	Wenn mehr als 7 Bewegungsabläufe nacheinander aufgezeichnet und gespeichert werden, so kommt es teilweise vor, dass sie nicht mehr korrekt in der Liste angezeigt werden. Dann muss man die Applikation neustarten um alle Einträge korrekt anzuzeigen. Das Problem liegt irgendwo bei der Kommunikation zwischen C# und C++ Code, konnte aber noch nicht behoben werden.
2.13	Feature nicht implementiert

Tabelle 2: Nicht erfolgreiche Testfälle auf Roboter Controller

3 Systemtests Remote Controller

3.1 Testfälle

Id	Beschreibung	Erwartetes Resultat	Requirement	OK?
2.1	Nach verfügbaren Roboter Controllern suchen	Mit einem Klick auf den „Discovery“ Button wird der DPWS Probe Vorgang gestartet und anschliessend alle Roboter Controller angezeigt, die im Netz verfügbar sind.	R1.2, R13.1	OK
2.2	Mit Roboter Controller verbinden	Nach Auswahl eines gefundenen Roboter Controllers und Klick auf den „Connect“ Button werden die Metadaten ausgetauscht, die Verbindung zum Roboter Controller hergestellt und alle vorhandenen Events abonniert.	R1.1, R1.3, R1.4, R13.1	OK
2.3	Roboter steuern	Nach erfolgreichem Verbindungsaufbau können alle Achsen auf dieselbe Art vom Remote Controller gesteuert werden wie auf dem Roboter Controller.	R1.1, R13.2	OK
2.4	Robotersteuerung per Knopfdruck	Jegliche Roboter Aktion wird immer solange ausgeführt, wie ein Knopf nach unten gedrückt wird.	R1.1, R13.2	NOK
2.5	Bewegungsabläufe starten	Der Remote Controller kann sich alle Bewegungsabläufe anzeigen lassen und anschliessend einen beliebigen Ablauf auswählen und mit einem Klick auf „Play“ starten. Der Bewegungsablauf startet gleich wie wenn er vom Roboter Controller gestartet würde.	R1.1, R13.3	OK
2.6	Betriebsdaten anzeigen	Alle Betriebsdaten des Roboters werden auf dem Display des Remote Controllers korrekt dargestellt.	R1.1, R14.1	OK
2.7	Fehlerzustände anzeigen	Erhält der Roboter Controller ein Fehler vom Roboter Treiber und leitet diesen weiter, so muss der Fehler sofort auf dem Display des Remote Controllers angezeigt werden.	R1.4, R15.1	OK
2.8	Alarme anzeigen	Löst der Roboter Controller ein Alarm aus, wird dieser sofort auf dem Display des Remote Controllers angezeigt.	R1.4, R16.1	OK
2.9	Notaus anzeigen	Löst der Roboter Controller ein Notaus aus, wird dieses sofort auf dem Display des Remote Controllers angezeigt. Der Remote Controller kann keine weitere Aktionen ausführen, bis der Notaus aufgelöst wurde.	R1.4, R17.1	OK
2.10	Notaus auslösen	Wird der Notaus Knopf auf dem Display gedrückt, so wird sofort ein Notaus ausgelöst und jegliche Aktionen des Roboters gestoppt.	R1.1, R1.4, R17.2	OK

Tabelle 3: Testfälle Remote Controller

3.2 Nicht erfolgreiche Testfälle

Id	Begründung
2.4	Im Normalfall klappt die Steuerung genau so, dass die Aktion nur solange ausgeführt wird, wie man auf die Taste drückt. Tippt man aber nur sehr kurz auf einen Knopf, so kann es teilweise vorkommen, dass die Aktion zwar gestartet aber nicht mehr gestoppt wird. Dies ist so, weil das Control aus noch ungeklärten Gründen kein MousUp Event erhält.

Tabelle 4: Nicht erfolgreiche Testfälle auf Remote Controller

4 Verzeichnisse

4.1 Tabellenverzeichnis

Quick Start Guides

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

Review

Datum	Kommentar	Revisor
12.06.2010	Review DPWS Entwicklungs Anleitung	TZ
13.06.2010	Review Roboter- & Remote Controller Anleitung	SZ
14.06.2010	Review Installationsanleitung	TZ
16.06.2010	Abschlussreview vor Druck	SZ

Tabelle 1: Reviews

Teil 1: Benutzeranleitung Roboter Controller

1 Screens vom Roboter Controller

Steuerung

- Steuerung des Roboters mit Buttons
- Steuerung über Tastatur möglich, Hotkeys in grau angezeigt
- Manuelles Notaus auslösen
- Tastaturbelegung kann in der Konfigurationsdatei `RC.RobotController.exe.config` angepasst werden.

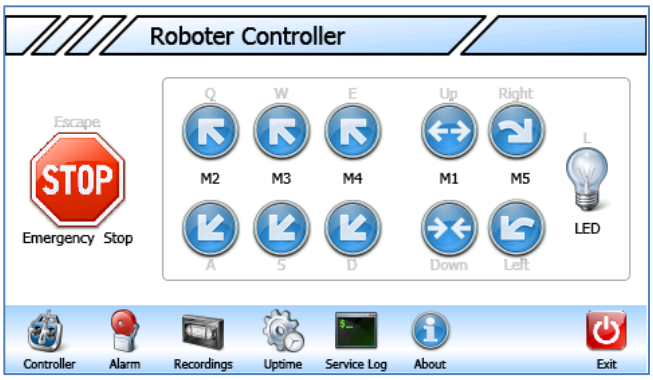


Abbildung 1: Roboter Controller Steuerung

Alarme

- Konfiguration der Alarme pro Achse
- Konfiguration Auto-Notaus

Auto-Notaus definiert die Dauer in Sekunden, wie lange ein Button ununterbrochen gedrückt werden darf, bis ein Notaus ausgelöst wird.

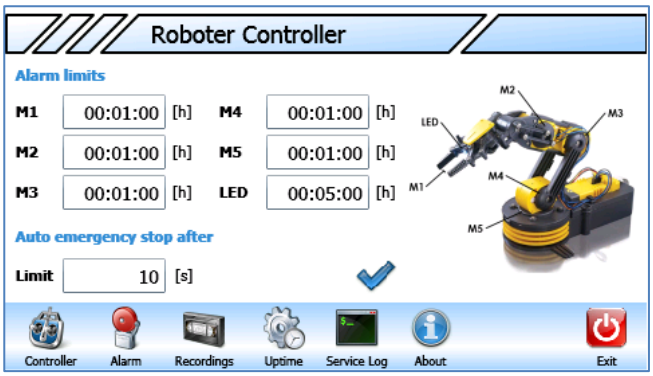


Abbildung 2: Alarmkonfiguration Screenshot

Bewegungsabläufe

- Auflistung aller Bewegungsabläufe
- Selektierter Bewegungsablauf kann vor- bzw. rückwärts abgespielt oder gelöscht werden
- Abspielender Bewegungsablauf kann gestoppt werden
- Neue Aufnahme starten: Das UI wechselt auf den Steuerungsscreen

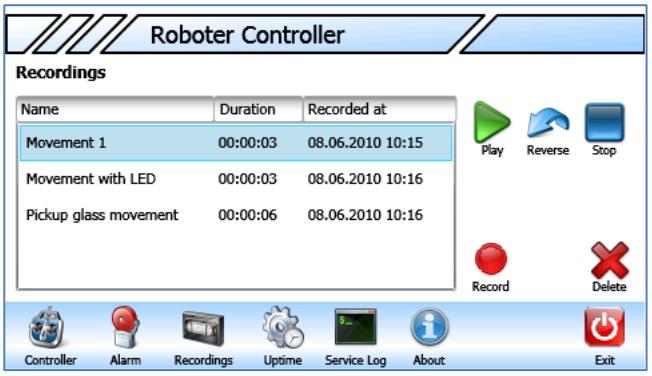


Abbildung 3: Aufnahmen Screenshot

Betriebszeit

Auf diesem Screen werden alle Betriebszeiten der Achsen/LED angezeigt

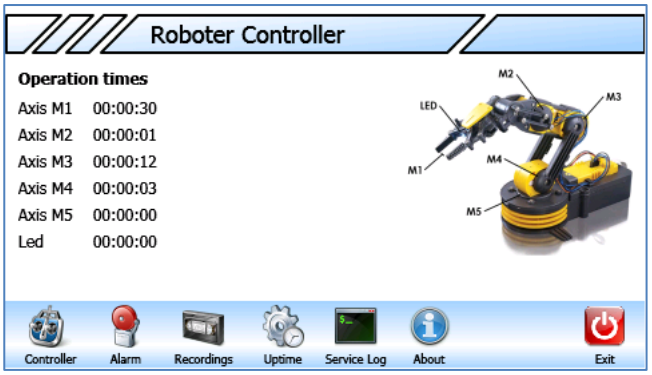


Abbildung 4: Screenshot Betriebszeiten

Service Log

Diverse Ereignisse wie zum Beispiel Fehlermeldungen, Notaus, Quittierung eines Notaus oder Alarmüberschreitungen werden in ein Service Log geschrieben. Dieses Service Log kann hier angeschaut werden. Zusätzlich zum Ereignis wird das Datum angezeigt.

Das Service Log wird im Dateisystem in der Datei `servicelog.txt` gespeichert.

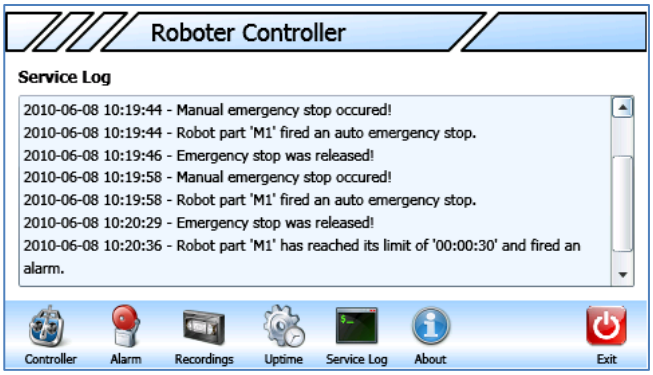


Abbildung 5: Alarmkonfiguration Screenshot

2 Applikationskonfiguration

In der Applikationskonfigurationsdatei (`RC.RobotController.exe.config`) kann die Tastaturbelegung, Alarmlimit, Auto-Notaus und DPWS Device Informationen konfiguriert werden. Die XML Datei ist weitgehend selbsterklärend.

Jedes Gerät benötigt eine eindeutige `DeviceEndpointAddress`. Wenn im XML keine `DeviceEndpointAddress` definiert ist, wird die Anwendung selber eine GUID generieren und diese im XML speichern.

Des Weiteren kann unter `<log4net>` das Logging konfiguriert werden. Als Loggingframework wird Apache log4net verwendet *[URL7.1]*.

Teil 2: Benutzeranleitung Remote Controller

3 Screens vom Remote Controller

Folgende Tabelle erklärt kurz und prägnant alle Masken vom Remote Controller.

Discovery

- Sichtbare Geräte werden in der Tabelle aufgelistet
- Mit Connect kann auf den selektierten Roboter Controller verbunden werden
- Beim Verbinden wird ein Pairing vorgenommen, so dass gleichzeitig nur ein Remote Controller zu einem Roboter Controller verbunden sein kann.

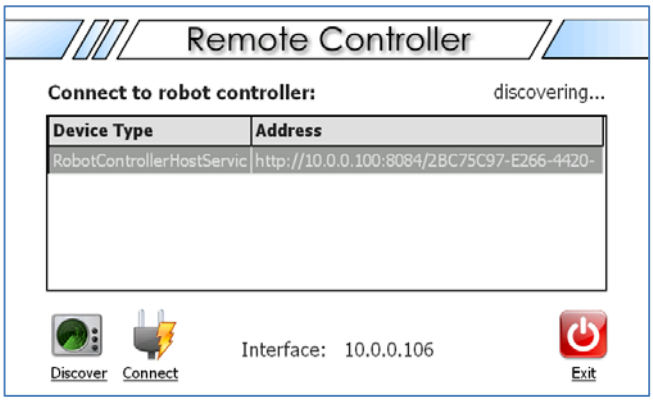


Abbildung 6: Remote Controller Discovery

Steuerung

- Navigation zu Bewegungsabläufe (Recordings), Betriebsdaten anzeigen (Uptime) und About Screen
- Mithilfe der Buttons kann der Roboter gesteuert werden oder ein Notaus ausgelöst werden
- Verbindung zum Roboter Controller trennen

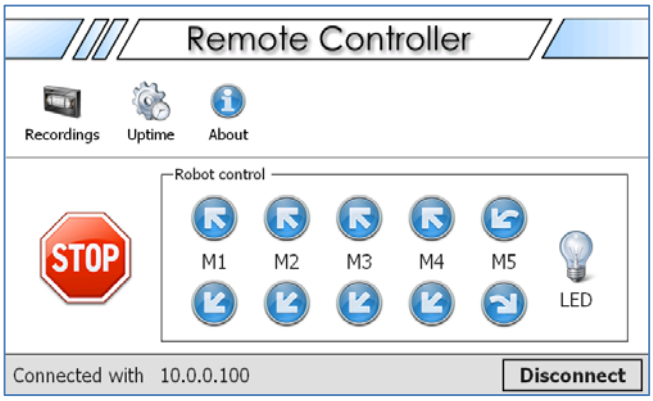


Abbildung 7: Remote Controller Steuerung

Alarmmeldung

Diese Meldung wird bei einem Alarm angezeigt. Die Alarme sind lediglich informativ und blockieren daher nichts. Sie werden auch nicht ins Roboternetz weitergeleitet.

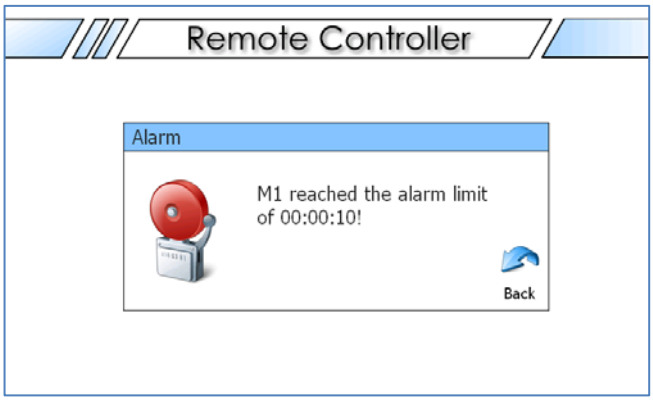


Abbildung 8: Remote Controller Alarmmeldung

Notaus Meldung

Falls ein Notaus ausgelöst wurde, wird der Bildschirm mit dieser Meldung gesperrt. Der Bildschirm blockiert, d.h. er geht erst Weg, wenn das Notaus quittiert wurde. Es besteht jedoch die Möglichkeit die Verbindung zum Roboter Controller zu trennen.

Abbildung 9: Remote Controller Notaus Meldung

Betriebszeit

- Anzeige der Betriebsdaten der einzelnen Achsen/LED

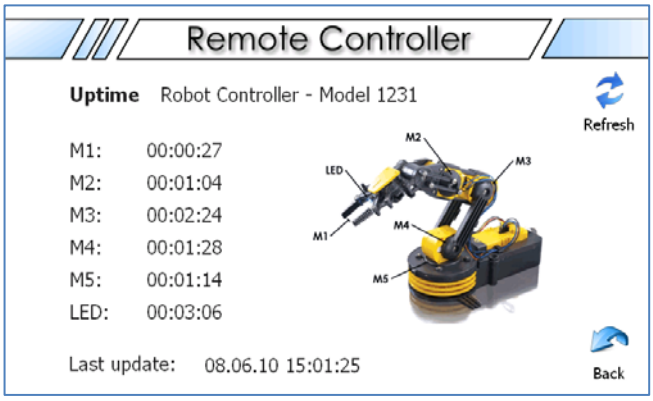


Abbildung 10: Remote Controller Betriebsdaten

Bewegungsabläufe

- Liste aller Bewegungsabläufe auf Roboter Controller
- Markierter Bewegungsablauf kann abgespielt oder gestoppt werden

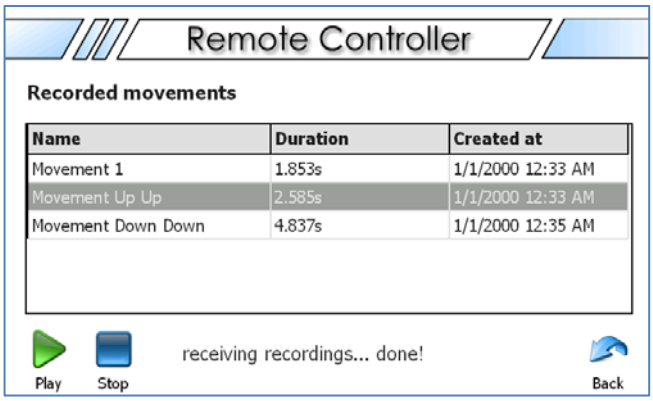


Abbildung 11: Remote Controller Betriebsdaten

4 Applikationskonfiguration

Neben der log4net Konfiguration unter `<log4net>` kann der `Endpoint` konfiguriert werden. Der `Endpoint` muss einmalig sein. Wenn der Wert leer ist, wird von der Applikation automatisch eine GUID generiert und im XML gespeichert.

Teil 3: Installationsanleitung Entwicklungsumgebung

5 Installation für Windows Embedded CE 6.0 Entwicklung

Folgende Schritte erklären den Installationsprozess um eine Entwicklungsumgebung für Windows Embedded CE 6.0 R3 aufzusetzen. Hat man für das Ziel-Device bereits ein SDK zur Verfügung, so können die Schritte 1-4 ausgelassen werden. Visual Studio 2005 wird nur noch gebraucht um mit dem Platform Builder ein für das Embedded Device abgestimmtes OSDesign und SDK zu erstellen.

(1) Visual Studio 2005 Professional

a) Installiere Visual Studio 2005
wird benötigt um ein OSDesign und zugehörendendes SDK zu erstellen

b) Installiere Visual Studio 2005 Team Suite Service Pack 1
http://www.microsoft.com/downloads/details.aspx?FamilyId=BB4A75AB-E2D4-4C96-B39D-37BAF6B5B1DC

c) Installiere Visual Studio 2005 Service Pack 1 Update for Windows Vista
wird benötigt bei Windows Vista / Windows 7
http://www.microsoft.com/downloads/details.aspx?FamilyID=90e2942d-3ad1-4873-a2ee-4acc0aace5b6

(2) Windows Embedded CE 6.0

a) Installiere Windows Embedded CE 6.0

b) Installiere Windows Embedded CE 6.0 Platform Builder Service Pack 1
http://www.microsoft.com/downloads/details.aspx?FamilyId=BF0DC0E3-8575-4860-A8E3-290ADF242678

(3) Windows Embedded CE 6.0 R2

a) Installiere Windows Embedded CE 6.0 R2

(4) Windows Embedded CE 6.0 R3

a) Installiere Windows Embedded CE 6.0 R3

b) Installiere Windows Embedded CE 6.0 R3 2009 Update Rollup
http://www.microsoft.com/downloads/details.aspx?FamilyID=18a0bc2b-35ad-4b04-bf80-3eb701ceaa2b

c) Installiere Windows Embedded CE 6.0 R3 2010 Monthly Update January
http://www.microsoft.com/downloads/details.aspx?FamilyID=0a003b66-5c00-43c2-8658-5453be22c402

d) Installiere Windows Embedded CE 6.0 R3 2010 Monthly Update February
http://www.microsoft.com/downloads/details.aspx?FamilyID=ba593dec-28a7-4558-aafc-ae85a14f74ae

e) Installiere Windows Embedded CE 6.0 R3 2010 Monthly Update March
http://www.microsoft.com/downloads/details.aspx?FamilyID=4dce5160-b049-417c-a0e3-c188939133dc

(5) Visual Studio 2008 Professional

a) Installiere Visual Studio 2008

(6) Expression Blend 2

a) (Optional) Installiere Expression Blend 2
für Design von Silverlight for embedded Applikationen

b) Installiere Expression Blend 2 Service Pack 1
http://www.microsoft.com/downloads/details.aspx?FamilyId=EB9B5C48-BA2B-4C39-A1C3-135C60BBBE66

6 Installation für Windows Mobile 6.5 Entwicklung

Folgende Schritte erklären den Installationsprozess um eine Entwicklungsumgebung für Windows Mobile 6.5 aufzusetzen.

(1) Visual Studio 2008 Professional

a) Installiere Visual Studio 2008

(2) Windows Mobile Device Center

a) (Optional) Installiere Windows Mobile Device Center
http://www.microsoft.com/windowsmobile/en-us/downloads/microsoft/device-center-download.mspx

(3) Windows Mobile 6 Professional SDK Refresh

a) Installiere Windows Mobile 6 Professional SDK Refresh
http://www.microsoft.com/downloads/details.aspx?FamilyID=06111a3a-a651-4745-88ef-3d48091a390b

(4) Windows Mobile 6.5 Professional Developer Tool Kit

a) Installiere Windows Mobile 6.5 Professional Developer Tool Kit
http://www.microsoft.com/downloads/details.aspx?FamilyID=20686a1d-97a8-4f80-bc6a-ae010e085a6e

Teil 4: DPWS Entwicklung

7 Einleitung DPWS Entwicklung

Als Einführung in die DPWS (Devices Profile for Web Services *[OASIS]*) Entwicklung wird jedem empfohlen, das DPWS Kapitel im Buch *Expert .NET Micro Framework* von Jens Kühner *[KÜHNER08]* zu lesen. Es handelt sich dabei zwar um die DPWS Entwicklung im Micro Framework, doch der neue DPWS Code im Compact Framework ist beinahe identisch, da der Code aus dem Micro Framework portiert wurde.

Das folgende Kurztutorial zur DPWS Entwicklung ist stark an das Buch von Jens Kühner angelehnt. Es ist so gegliedert, wie man bei der Entwicklung einer DPWS Applikation gewöhnlich vorgeht:

1. (Installation DPWS Generator Add-in)
2. Generierung der Proxys/Stubs
3. DPWS Device
4. Discovery
5. Metadata Exchange
6. Messaging
7. Eventing

Für die folgenden Code Beispiele muss beachtet werden, dass das Client- wie auch das Device Projekt jeweils eine Referenz auf das `Dpws.Core.dll` Assembly haben müssen. Zusätzlich muss das Client Projekt das Assembly `Dpws.Client.dll` und das Device Projekt `Dpws.Device.dll` referenzieren.

8 DPWS Generator

8.1 Installation DPWS Generator Add-In

Bevor mit der Entwicklung losgelegt werden kann, wird empfohlen das DPWS Generator Add-In zu installieren. Der DPWS Code kann zwar auch vollständig mit dem Command-Line Utility `Dpws.SvcUtil.exe` generiert werden, es ist jedoch komfortabler dies direkt in der IDE auszuführen.

Die Setup Datei heisst `DpwsAddInSetup.msi` und kann ohne weitere Einstellungen installiert werden. Das Add-In wird anschliessend in das Verzeichnis `%APPDATA%\Microsoft\MSEnvShared\Addins` installiert.

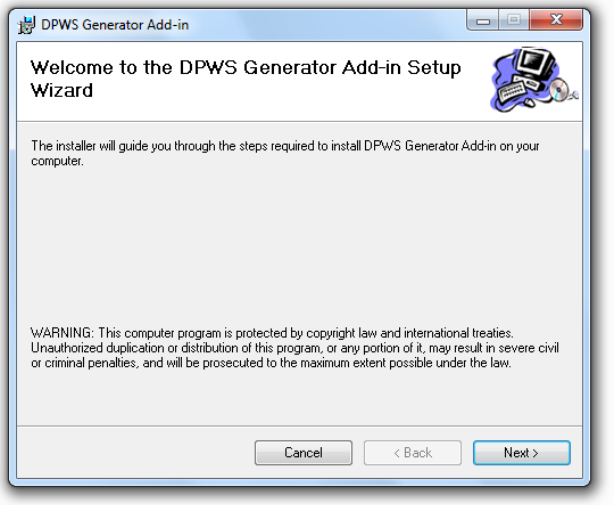


Abbildung 12: Setup DPWS Add-In

8.2 Dpws.SvcUtil

Das vom .NET Micro Framework portierte Codegenerierungs-Tool `Dpws.SvcUtil.exe` kann auf zwei Arten aufgerufen werden:

Generierung von Proxys/Stubs anhand einer WSDL Datei

```
Dpws.SvcUtil <wsdl file> [/D:DestinationDirectory] [/NS:Namespace]
             [/O:[ContractFilename]] [/V]
```

Parameter	Beschreibung
<wsdl file>	Pfad zur WSDL Datei. Diese Datei muss eine .wsdl Dateiendung besitzen.
/D:	Zielordner, wohin die generierten Dateien kopiert werden
/NS:	.NET Namespace der generierten Klassen. Wenn dieser Parameter weggelassen wird, so entspricht wird der angegebene TargetNamespace im WSDL verwendet.
/O:	Contract Source File Name. Der Name wird als Dateiname für den Contract Source und als Prefix für die Hosted Service und die Client Proxy Datei verwendet.
/V	Falls angegeben, werden erweiterte Informationen während dem Generieren angezeigt

Generierung einer WSDL Datei anhand eines Assembly

```
Dpws.SvcUtil <assembly file> [/D:DestinationDirectory] [/O:[WsdlFilename]]
             [/R:ReferencesPath] [/V]
```

Parameter	Beschreibung
<assembly file>	Pfad zum Assembly mit dem attributierten ServiceContract Interface. Diese Datei muss eine .exe oder .dll Dateiendung besitzen.
/D:	Zielordner, wohin das generierte WSDL kopiert werden soll
/O:	Dateiname der zu generierenden WSDL Datei
/R:	Pfad zu benötigten Referenzen, von denen das Inputassembly abhängig ist um geladen zu werden
/V	Falls angegeben, werden erweiterte Informationen während dem Generieren angezeigt

8.3 Integration in MSBuild

Der portierte DPWS Generator kann wie folgt in den MSBuild Prozess integriert werden:

```
<PropertyGroup>
    <DpwsGeneratorDir>$(SoloutionDir)Tools</DpwsGeneratorDir>
</PropertyGroup>
<PropertyGroup>
    <DpwsGeneratorTempDir>C:\Temp\DpwsGeneratedFiles\</DpwsGeneratorTempDir>
</PropertyGroup>
<ItemGroup>
    <GeneratedSourceFiles Include="$(DpwsGeneratorTempDir)*.*"/>
</ItemGroup>

<Target Name="BeforeBuild">
    <RemoveDir Directories="$(DpwsGeneratorTempDir)" />
    <MakeDir Directories="$(DpwsGeneratorTempDir)" />
    <Exec Command="$(DpwsGeneratorDir)Dpws.SvcUtil.exe
                   $(ProjectDir)ServiceContract\RobotControllerService.wsdl
                   /D:$(DpwsGeneratorTempDir) /O:RobotControllerService
                   /NS:RC.Common.ServiceContract" />
    <Copy SourceFiles="@(GeneratedSourceFiles)"
          DestinationFolder="$(ProjectDir)ServiceContract" OverwriteReadOnlyFiles="true" />
</Target>
```

9 Generierung der Proxy/Stubs

9.1 Generierung anhand WSDL

Nachdem man ein WSDL erstellt hat, können im Visual Studio per Rechtsklick auf die WSDL Datei „Generate DPWS Proxies & Stubs“ die Proxys und Stubs generiert werden.

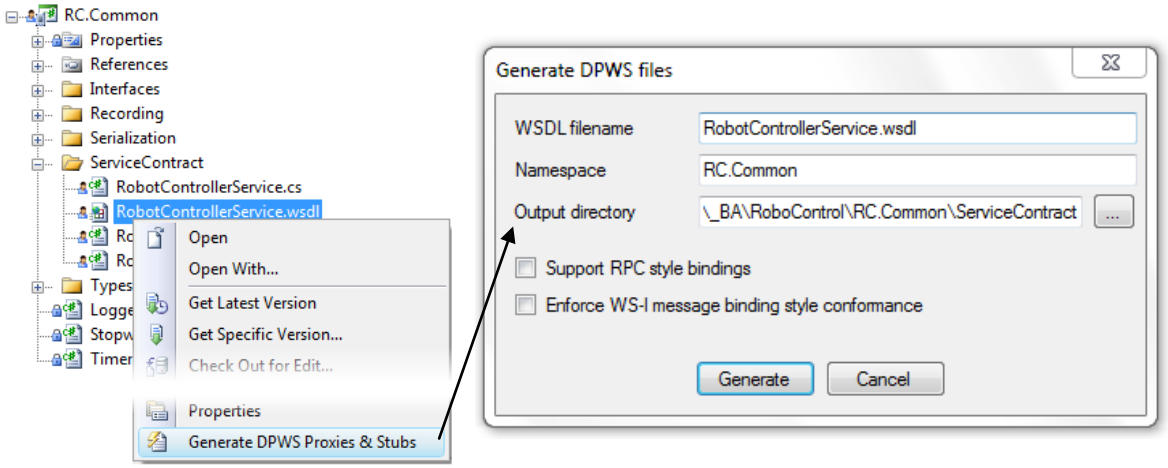


Abbildung 13: Screenshot Visual Studio Add-in

Falls das Projekt unter Source Control steht und als Output Directory direkt ein Ordner im Projekt angegeben ist, so sollten die bereits eingecheckten, generierten Dateien zuerst ausgecheckt werden, damit diese nach dem Überschreiben wieder korrekt eingecheckt werden können. Ansonsten werden die generierten Dateien nicht korrekt als „pending changes“ erkannt.

9.2 Generierung anhand Assembly

Möchte man ein Interface als Service Contract mit den passenden Attributen versehen und anhand des Assembly die Proxys/Stubs generieren, so muss man im Projekt eine Referenz auf das Assembly `Dpws.Core.dll` hinzufügen und dort die Attribute aus dem Namespace `Ws.ServiceModel` verwenden. Es handelt sich dabei um dieselben Attribute wie in WCF (z.b. `ServiceContract`, `OperationContract`, `DataContract`, etc.).

Zum Generieren geht man gleich vor wie bei der WSDL Datei. Hier wird dann aus dem Interface zuerst ein WSDL generiert, welches anschliessend weiterverwendet wird um die

Proxies/Stubs zu generieren. Dabei wird das Projekt kompiliert, weshalb keine Compilefehler vorhanden sein dürfen. Fehlermeldungen erscheinen wie gewohnt im Output Window.

10 DPWS Device Implementierung

10.1 Endpoint Adressen vs. Transport Adressen

Bevor mit der DPWS Device Implementierung begonnen werden kann, muss der Unterschied zwischen einer Endpoint Adresse und einer Transport Adresse geklärt werden. Eine Endpoint Adresse (oft auch logische Adresse genannt) ist eine eindeutige transportneutrale Adresse um Services und Messages zu adressieren. Eine Endpoint Adresse ist stets ein *Globally Unique Identifier* (GUID). Ein Beispiel einer Endpoint Adresse sieht folgendermassen aus:

`urn:uuid:2BC75A97-E266-4420-A66C-C7BD5EE5219A`

Die Transport Adresse (oft auch physische Adresse genannt) bestimmt wie ein Gerät erreicht werden kann. Normalerweise besteht diese aus einer IP, Port und der Endpoint Adresse des Geräts. Eine Transport Adresse kann folgendermassen aussehen:

`http://10.0.0.100:8084/2BC75A97-E266-4420-A66C-C7BD5EE5219A`

10.2 Host Services vs. Hosted Services

Neben dem Unterschied der Endpoint und der Transport Adresse muss auch der Unterschieder zwischen Host Services und Hosted Services geklärt werden.

Jedes DPWS Device kann null oder ein Host Service und null oder mehrere Hosted Services besitzen. Der Host Service kann als Hauptservice des Geräts angesehen werden, welcher direkt mit einem Device assoziiert wird. Zusätzlich können weitere spezifische Services als Hosted Services hinzugefügt werden. Hat man auf einem Gerät nur ein Service, so kann dieser direkt als Host Service gesetzt werden.

Neben den eigenen optionalen Hosted Services definiert der DPWS Standard drei weitere Hosted Services, welche jedes DPWS Device besitzt. Diese Hosted Services sind für Discovery, MetadataExchange und Eventing bestimmt.

Wie wir später beim Discovery sehen werden, spielt es beim Proben keine Rolle ob ein Service ein Host oder Hosted Service ist. Der einzige Unterschied liegt darin, dass ein `ProbeMatch` lediglich die Informationen zum Host Service liefert.

10.3 DPWS Device

Für die DPWS Device Implementierung nehmen wir als Grundlage den ProtoypeService, welcher im mitgelieferten Prototyp im Projekt `Prototype.ServiceContract` definiert ist. Dieser Service hat eine `OneWay` und eine `TwoWay` Methode und zusätzlich einen `EmergencyStop` Event.

Aus dem PrototypeService WSDL wurde ein `IPrototypeService` Interface und eine `PrototypeService` Hosted Service Klasse generiert. Der Hosted Service `PrototypeService` kann nun dem DPWS Device direkt als Host Service oder auch als Hosted Service hinzugefügt werden.

Zur Instanzierung der `PrototypeService` Hosted Service Klasse muss eine Instanz mitgegeben werden, die das Interface `IPrototypeService` implementiert. Aus diesem Grund erstellen wir eine `PrototypeServiceImpl` Klasse:

```
internal class PrototypeServiceImpl : IPrototypeService
{
    public void OneWay(OneWay req)
    {
        // do something with req.Param
    }

    public TwoWayResponse TwoWay(TwoWayRequest req)
    {
        return new TwoWayResponse {Sum = req.X + req.Y};
    }
}
```

Mit der erstellten `PrototypeServiceImpl` Klasse ist nun alles vorhanden, was gebraucht wird um den `PrototypeService` zu starten.

```
private static void DpwsStart()
{
    Console.WriteLine("Starting DPWS stack");

    // Set device information
    Device.EndpointAddress = "urn:uuid:cf16db78-02c9-c8ca-b37b-0000004071f6";
    Device.ThisModel.Manufacturer = "HSR";
    Device.ThisModel.ManufacturerUrl = "http://www.hsr.ch";
    Device.ThisDevice.FriendlyName = "HSR Prototype Device";

    // Add a Host service type for demonstration (host service has no service operations)
    var hostService = new DpwsHostedService();
    hostService.ServiceTypeName = "PrototypeServiceHost";
    hostService.ServiceNamespace = new WsXmlNamespace("pro",
                                        "http://schemas.hsr.ch/PrototypeServiceHost");
    hostService.ServiceId = "urn:uuid:cf16db78-02c9-c8ca-b37b-0000004071f6";

    Device.Host = hostService;

    // Add Dpws hosted service(s) to the device
    var hostedService = new PrototypeService(new PrototypeServiceImpl());
    Device.HostedServices.Add(hostedService);

    // Set this device property if you want to ignore this clients request
    Device.IgnoreLocalClientRequest = false;

    // Start the device
    Device.Start();
}
```

11 Discovery

Nachdem das Device gestartet wurde, können die DPWS Clients mit dem Discovery Prozess beginnen.

11.1 Hello und Bye Nachrichten empfangen

Da jedes Device gemäss dem WS-Discovery Standard Hello und Bye Nachrichten versendet, können alle Clients im LAN Segment diese Nachrichten empfangen und darauf reagieren.

```
using (var client = new PrototypeServiceClientProxy()) // initializing
{
    // Set this client property if you want to ignore this devices request
    client.IgnoreRequestFromThisIP = false;
    client.HelloEvent += new HelloEventHandler(client_HelloEvent);
    client.ByeEvent += new ByeEventHandler(client_ByeEvent);

    // further client code
    ...
}

private static void client_ByeEvent(object sender, DpwsServiceDescription e)
{
    // bye event occurred
}

private static void client_HelloEvent(object sender, DpwsServiceDescription e)
{
    // hello event occurred
}
```

Achtung: Die Hello und Bye Event Handler vom DPWS Stack werden in einem separaten Thread aufgerufen. Darum müssen die Zugriffe auf das UI über Control.Invoke() auf den UI Thread dispatched werden.

11.2 Probing Prozess

Um explizit nach einem Gerättyp im LAN Segment zu suchen kann ein Client gemäss WS-Discovery Standard eine Probe Nachricht per Multicast ins Netz senden, worauf alle passenden Geräte eine Probe Match Nachricht zurücksenden.

```
using (var client = new PrototypeServiceClientProxy()) // initializing
{
    DpwsServiceTypes types = new DpwsServiceTypes();
    types.Add(new DpwsServiceType("PrototypeService",
                                  "http://schemas.hsr.ch/PrototypeService"));
    DpwsServiceDescriptions probeMatches = null;

    try
    {
        probeMatches = client.DiscoveryClient.Probe(types);

        if (probeMatches != null)
            Console.WriteLine("ProbeMatches.Count: " + probeMatches.Count);
        else
            Console.WriteLine("ProbeMatches.Count: 0 (object was null)");
    }
    catch (Exception e)
    {
        // exception handling
    }

    // further client code
    ...
}
```

Beim Probing kann nach Host Services oder Hosted Services gesucht werden. Egal nach was gesucht wird, die Antwort beinhaltet immer Informationen zum Host Service. Weitere Informationen müssen via Metadata Exchange angefordert werden.

Zusätzlich zum Probing via Multicast kann ein Client auch einen Directed Probe zu einer bestimmten Transport Adresse per Unicast ausführen.

12 Metadata Exchange

Um die Metadaten eines Devices abzufragen wird die Transport Adresse des Gerätes benötigt. Im Normalfall wird diese mit dem XAddr Parameter in der Probe Match Antwort mitgeliefert. Falls dies nicht der Fall ist, muss nach dem Probe Request ein zusätzlicher Resolve Request gesendet werden um anhand der Endpoint Adresse die Transport Adresse herauszufinden.

Wie bereits im Kapitel übers Probing erläutert, liefert eine Probe Match Response lediglich die Endpoint Adresse des Host Service. Vielfach möchte man aber Informationen und Endpoint Adressen von Hosted Services. Diese Informationen erlangt man über den Metadata Exchange:

```
DpwsServiceDescription dpwsServiceDesc = ... // retrieved via probing
string transportAddr = dpwsServiceDesc.XAddrs[0];
DpwsMetadata metadata = DpwsMexClient.Get(transportAddr);
```

Die Metadata Klasse liefert folgende Informationen zu einem Device:

- **ThisModel**: Gibt Informationen zum Gerätetyp. Informationen sind zum Beispiel Manufacturer, Modelname, Modelnummer
- **ThisDevice**: Gibt Informationen über das spezifische Gerät. Informationen sind zum Beispiel Friendly Name, Firmware Version, Seriennummer
- **Relationship**: Liefert eine Liste von allen Hosted Services auf dem Gerät. Zu einem Hosted Service wird jeweils die Service ID (Endpoint) und der Servicetyp geliefert.

```
namespace Dpws.Client.Mex
{
    /// <summary>
    /// Used to store service endpoint metadata details aquired from a Get metadata response.
    /// </summary>
    public class DpwsMetadata
    {
        public DpwsThisModel ThisModel { get; private set; }

        public DpwsThisDevice ThisDevice { get; private set; }

        public DpwsRelationship Relationship { get; private set; }
    }
}
```

13 Messaging

Nach dem Probing und Metadata Exchange ist der Client nun in der Lage die Web Services des Devices aufzurufen. Dank dem generierten Client Proxy ist dies nun kein Problem mehr. Für jede definierte Methode in der WSDL Datei/Interface wurde im Client Proxy eine Methode und passende Data Transfer Objects generiert. Der Client Code um die `OneWay` und die `TwoWay` Methode aufzurufen sieht folgendermassen aus:

```
using (var client = new PrototypeServiceClientProxy())
{
    // do probe

    // do resolve (optionally)

    // do metadata exchange

    client.ServiceEndpoint = // transport address of device
    client.OneWay(new OneWay { Param = 1 });

    TwoWayResponse res = client.TwoWayRequest(new TwoWayRequest { X = 1, Y = 2 });
    Console.WriteLine("TwoWay response is: " + res.Sum);
}
```

14 Eventing

14.1 Definition eines Events

14.1.1 Definition von Events im WSDL

Damit der DPWS Generator automatisch den Eventing Code bzw. die Event Sources für das Device erstellt, müssen die Event Methoden im WSDL speziell definiert werden. Als erstes muss beim WSDL `PortType` das Attribute `wse:EventSource="true"` gesetzt werden. Des Weiteren muss jede WSDL Operation so definiert werden, dass sie nur ein WSDL `Output` und kein `Input` Element besitzt.

```
<wsdl:portType name="PrototypeService" wse:EventSource="true">
    <wsdl:operation name="OneWay">
        <wsdl:input
            message="tns:OneWayMessageIn"
            wsa:Action="http://schemas.hsr.ch/PrototypeService/OneWay"/>
    </wsdl:operation>
    <wsdl:operation name="TwoWay">
        <wsdl:input
            message="tns:TwoWayMessageIn"
            wsa:Action="http://schemas.hsr.ch/PrototypeService/TwoWayRequest"/>
        <wsdl:output
            message="tns:TwoWayMessageOut"
            wsa:Action="http://schemas.hsr.ch/PrototypeService/TwoWayResponse"/>
    </wsdl:operation>
    <wsdl:operation name="EmergencyStop">
      <wsdl:output
          message="tns:EmergencyStopMessageOut"
          wsa:Action="http://schemas.hsr.ch/PrototypeService/EmergencyStop"/>
    </wsdl:operation>
</wsdl:portType>
```

14.1.2 Definition von Events über Service Contract Interface

Events können auch wie gewohnt bei WCF am Service Contract Interface mittels `CallbackContracts` attributiert werden.

```
using Ws.ServiceModel;

[ServiceContract(Namespace="http://schemas.hsr.ch/PrototypeService",
                 CallbackContract=typeof(IPrototypeServiceCallback))]
public interface IPrototypeService
{
    [OperationContract(Action="http://schemas.hsr.ch/PrototypeService/OneWay", IsOneWay=true)]
    void OneWay(OneWay req);

    [OperationContract(Action="http://schemas.hsr.ch/PrototypeService/TwoWayRequest")]
    TwoWayResponse TwoWay(TwoWayRequest req);
}

public interface IPrototypeServiceCallback
{
    [OperationContract(Action="http://schemas.hsr.ch/PrototypeService/EmergencyStop")]
    void EmergencyStop(EmergencyStopResponse resp);
}
```

14.2 Event auf Client

14.2.1 Callback Interface implementieren

Sobald ein Callback Interface existiert, muss der Client beim Instanzieren des Client Proxys eine Referenz auf eine Implementierung des Callback Interfaces mitgeben.

```
class PrototypeClient : IPrototypeServiceCallback
{
  public void StartClient()
  {
    using (var client = new PrototypeServiceClientProxy(this))
    {
      // do probe

      // do resolve (optionally)

      // do metadata exchange

      // ...
    }
  }

  #region IPrototypeEventsCallback Members

  public void EmergencyStop(EmergencyStopResponse resp)
  {
     ThreadPool.QueueUserWorkItem(o =>
                MessageBox.Show("Emergency-Stop Event received.", "Emergency!"));
  }

  #endregion
}
```

Achtung: Die Callback Methoden auf dem Client vom DPWS Stack werden synchron aufgerufen, weshalb längere Aktionen innerhalb der Callback Methode in einen Thread/Threadpool ausgelagert werden müssen, da ansonsten das Device in ein Timeout gerät, während es auf die Response des Clients wartet.

14.2.2 Event auf Client abonnieren

Folgendes Code Beispiel zeigt wie auf dem DPWS Client ein Event abonniert werden kann.

```
string transportAddress = // transport address of device

DpwsServiceType subscriptionType =
               new DpwsServiceType("EmergencyStop", "http://schemas.hsr.ch/PrototypeService");

DpwsSubscribeRequest request =
           new DpwsSubscribeRequest(subscriptionType, // subscription type
                                    transportAddress, // event source address
                                    _serviceProxy.TransportAddress,// client transport address
                                    null,                          // expires never
                                    null);

DpwsEventSubscription subscription = DpwsEventingClient.Subscribe(request);
```

Beim Abonnieren eines Events kann angegeben werden, wie lange das Abonnement gültig sein soll. Wird `null` mitgegeben, so läuft das Abonnement nie ab.

14.3 Events auslösen auf Device

Das DPWS Device kann zu jederzeit einen Event auslösen. Sofern es Clients gibt, welche den Event abonniert haben, so werden diese über Unicasts vom Device benachrichtigt. Der generierte Hosted Service enthält für jeden Event eine entsprechende Methode.

```
PrototypeService hostedService= // hosted service which was added to Device.HostedServices

hostedService.EmergencyStop(new EmergencyStopResponse {ErrorCode = 1});
```

Teil 5: Anhang

15 Verzeichnisse

15.1 Tabellenverzeichnis

15.2 Abbildungsverzeichnis

15.3 Literaturverzeichnis

[Kühner08] Kühner, J 2008 Expert .NET Micro Framework, Apress
http://www.apress.com/book/view/159059973x

[OASIS] OASIS Spezifikation
http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html

[Url7.1] Offizielle Apache log4net Webpräsenz
http://logging.apache.org/log4net/

Alle Weblinks sind im Juni 2010 auf Ihre Erreichbarkeit überprüft worden.

Präsentationen

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

1 Zwischenpräsentation vom 14.04.2010

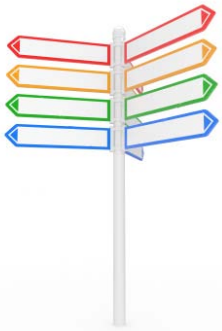
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Bachelorarbeit: Roboter Controller

Stefan Züger
Tobias Zürcher

Zwischenpräsentation vom 14.04.2010

Inhaltsverzeichnis



Auftraggeber & Betreuer

- Betreuer
  - Prof. Hansjörg Huser
  - Dipl. Ing. Jürg Jucker
- Auftraggeber: Zühlke Engineering AG
  - Dipl. Ing. Mario Klessascheck
    - Business Unit Leiter
  - Dipl. Ing. Michael Koster:
    - Windows Embedded/Mobile Experte, MVP (technische Konsultation)

ins institute for networked solutions

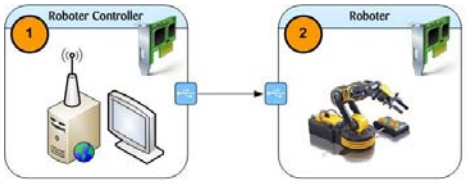
zühlke empowering ideas

Aufgabenstellung

- Schulungsplattform aufbauen
  - Für interne Schulungen im Bereich Embedded Entwicklung
- Testen neuer
  - Kommunikationskonzepte
    - «Managed Webserver» zur Remotesteuerung eines Roboters
  - GUI Technologien
    - Silverlight for Embedded

Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

Show-Case Anwendung



1 **Roboter Controller**
Windows Embedded CE 6.0 R3
CF 3.5

2 **Roboter**
nativer Treiber (von Zühlke entwickelt)

3 **Remote Controller**
Windows Mobile 6.5 (CF 3.5)

Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

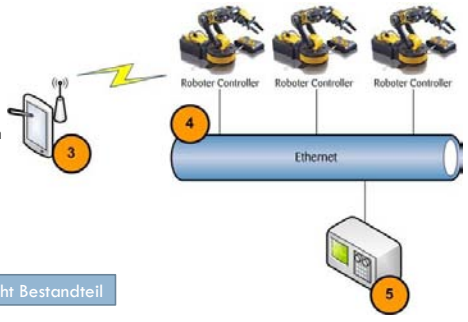
Show-Case Anwendung

3 **Remote Controller**

4 **Roboternetz**
- Wartungsarbeiten
- Not-Aus

5 **Monitoring Zentrale**

Nicht Bestandteil



Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

Kommunikationskonzept

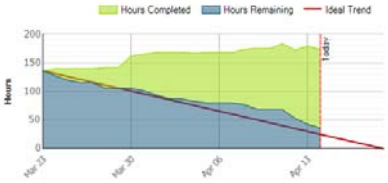
- Integration in Visual Studio analog WCF
  - Anhand Interface WSDL & Proxys generieren
    - Und umgekehrt

Ziele zusammengefasst

- Managed Implementation DPWS-Stack
- Touchinterface: Silverlight for embedded
- „Show-Case“ als Schulungsplattform
  - Beispielimplementation
  - Tutorial für Schulung

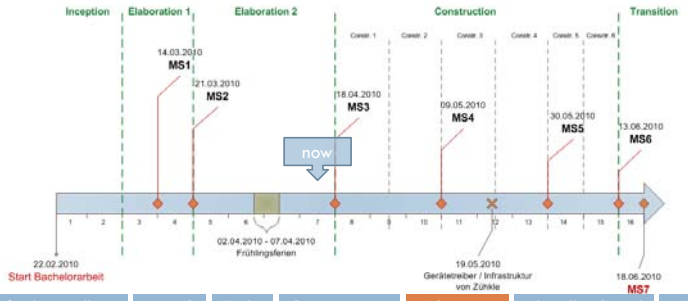
Organisation

- Team Foundation Server 2010
  - Taskmanagement (Zeitbuchungen auf Tasks)
  - Source Control Management
  - Continuous Integration
  - Test-Case Management
    - Code Coverage
  - Teamportal
    - Wiki
    - Reporting
    - Informationsportal
  - Sharepoint



Planung: Timeline/Milestones

- MS1: Projektplan genehmigt
- MS2: Requirement Specification genehmigt
- MS3: Systemarchitektur geführt, Entwicklungsstop



Aufgabenstellung | DPWS | Ziele | Organisation | Planung | Aktueller Stand | Fragen

Schwierigkeiten & Herausforderungen

- Installationsmarathon
- Viele neue Technologien
  - Windows Embedded CE
  - Windows Mobile
  - Compact Framework
- Silverlight for embedded ist **native (C++)** ☹
  - Kommunikation über Callbacks mit Funktionspointer, Pinvoke, fehlende Doku, unpräzise Fehlermeldungen
- Emulation für Test & Debugging problematisch
  - Multicastempfang nicht unterstützt bei Emulatoren

Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

Aktueller Stand & Ausblick

- Erreicht
  - DPWS-Stack aus Micro-Framework portiert
  - Hardware Testumgebung eingerichtet
  - Technologiedurchstoss gelungen
- Nächste Schritte
  - Präsentation des Prototypen bei Zühlke
  - Refactoring der DPWS Portierung
  - Implementation der Features gemäss SRS

Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

Fragen

- Fragen?
- Falls keine mehr: wir haben auch Fragen ☺

Aufgabenstellung | DPWS | Ziele | Organisation | Milestones | Aktueller Stand | Fragen

2 Präsentation des Prototyps vom 21.04.2010

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Milestone 3:
DPWS & Silverlight „Proof of concept“

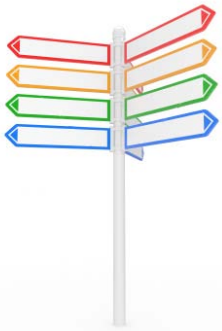
Stefan Züger
Tobias Zürcher

Präsentation des Prototypen am 21.04.2010

Inhaltsverzeichnis

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- DPWS Implementation
  - Varianten, Entscheidung
  - Änderungen an MF-Code
- Silverlight for embedded
  - XAML2CPP
  - MVVM mit native/managed
- Demo Prototyp
- Probleme / Fragen



Index | DPWS Implementation | Silverlight for embedded | DPWS Demo | Probleme/Fragen

DPWS Implementation

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- Portierung Micro-Framework Code
  - Apache 2.0 Lizenz
- Anpassungen
  - Fehlende Klassen aus System.Net & System.Net.Sockets
    - InputNetworkStreamWriter, OutputNetworkStreamWriter, HttpListener, HttpListenerContext, HttpListenerRequest, HttpListenerResponse, NetworkStream
    - Aus Micro-Framework portiert
  - Verwendung der XML Writer/Reader aus CF
  - Debug & Info Outputs → Apache log4net

Index | DPWS Implementation | Silverlight for embedded | DPWS Demo | Probleme/Fragen

Tools aus MF

- MFSvcUtil.exe zur Generierung Stubs/Proxy aus WSDL
  - DataContract (Request/Response)
  - DataContractSerializer pro DataContract
  - Service-Interface
  - Callback-Interface für Eventing
  - ServiceClientProxy
  - PrototypeServiceClientProxy.cs
  - PrototypeServiceHostedService.cs

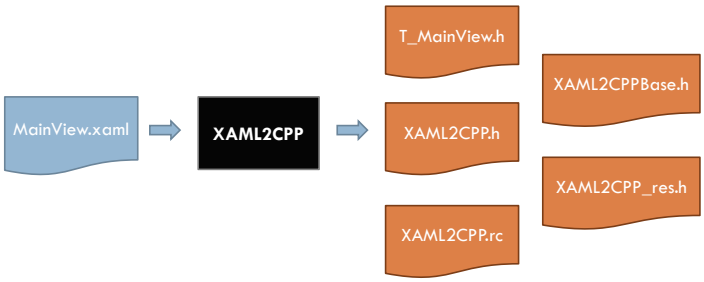
PrototypeService.cs

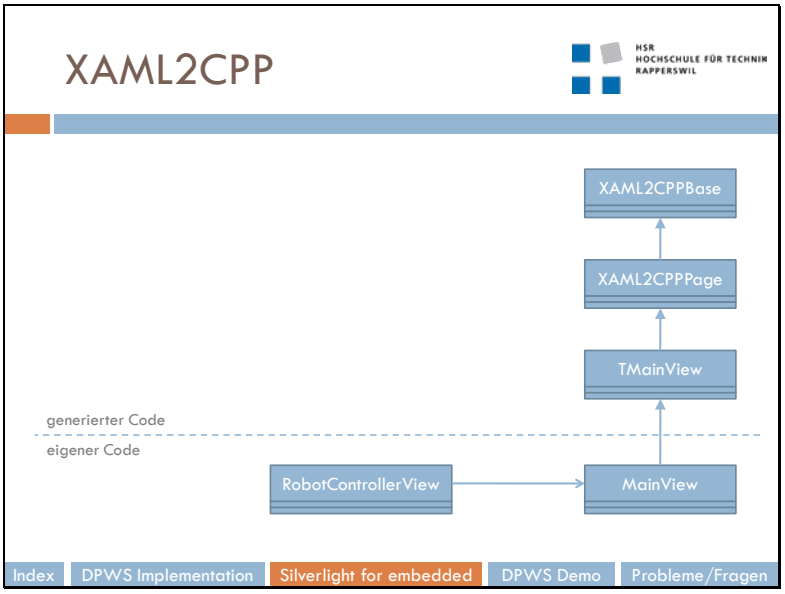
Silverlight for embedded

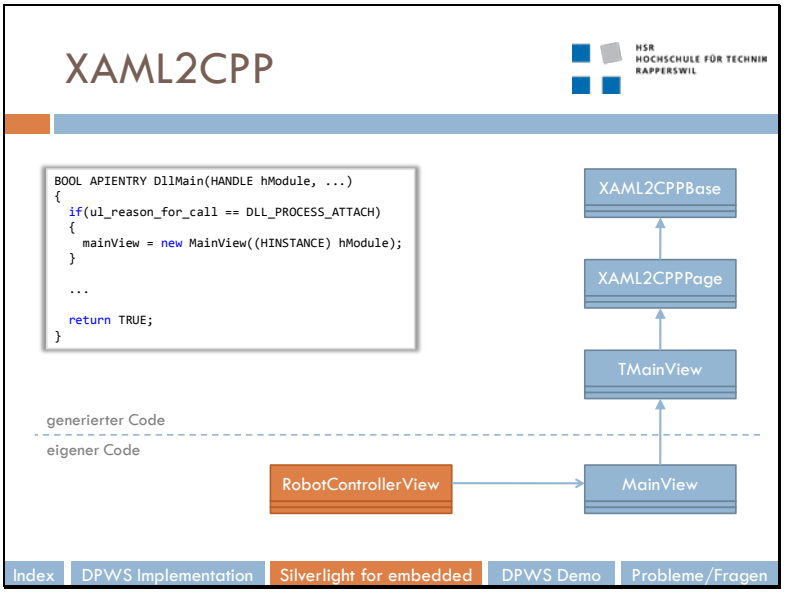
- Silverlight for embedded != Silverlight
- Code-behind C++
  - GUI Elemente manuell binden
  - Event-Handler manuell binden
- Storyboards funktionsfähig
- Sehr wenig Dokumentation ☹
  - Schlechtes Error-Handling

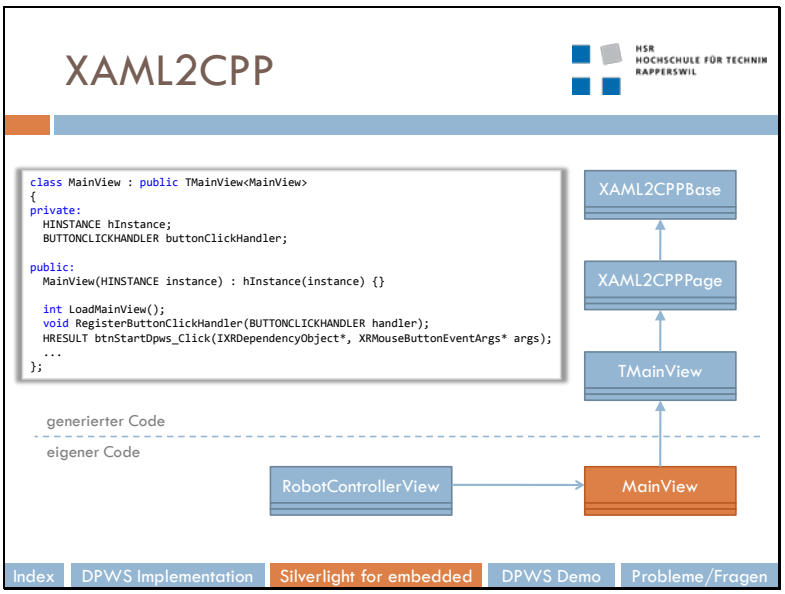
XAML2CPP

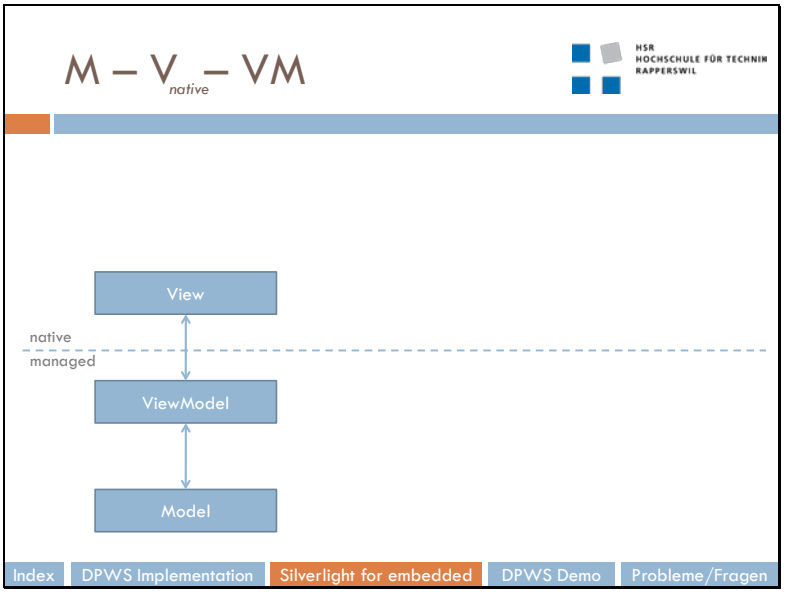
- Command-Line Utility von Valter Minute
  - Generiert C++ aus XAML File

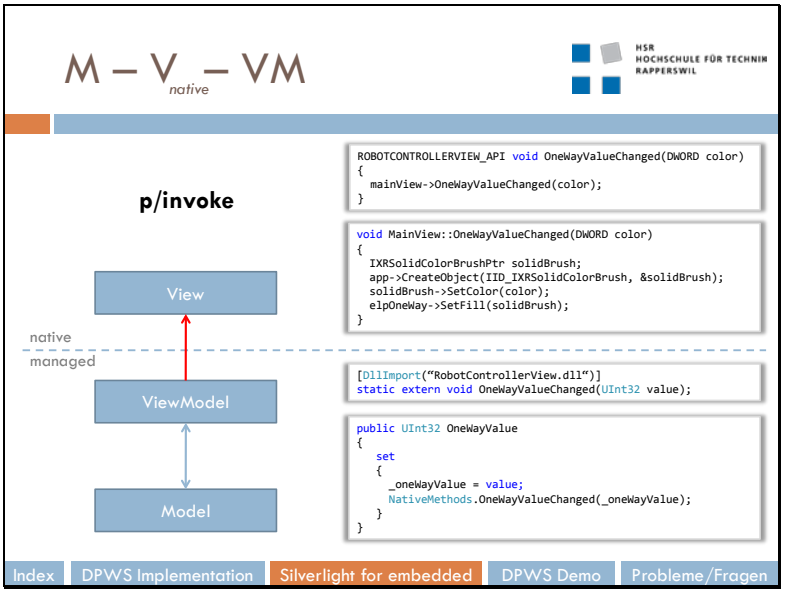


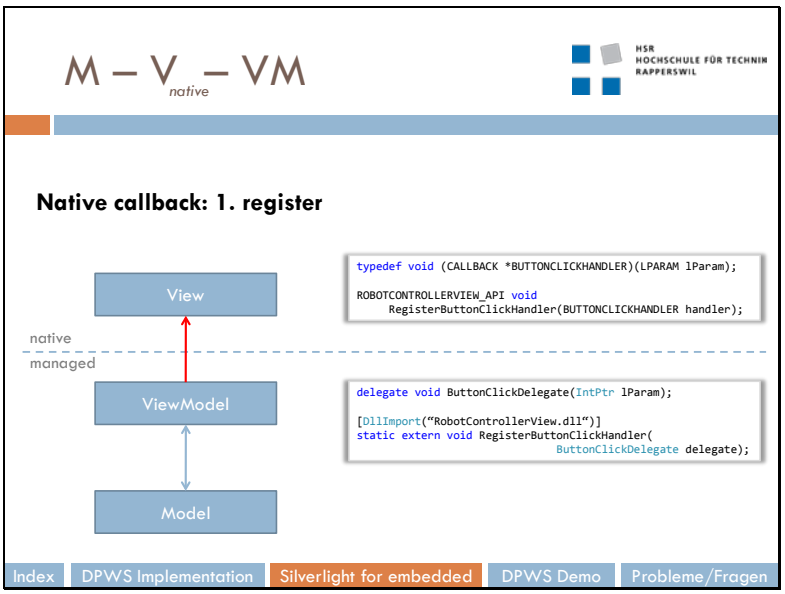

XAML2CPP
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
XAML2CPPBase
XAML2CPPPage
TMainView
generierter Code
eigener Code
RobotControllerView
MainView
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen


XAML2CPP
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
BOOL APIENTRY DllMain(HANDLE hModule, ...)
{
if(ul_reason_for_call == DLL_PROCESS_ATTACH)
{
mainView = new MainView((HINSTANCE) hModule);
}
...
return TRUE;
}
XAML2CPPBase
XAML2CPPPage
TMainView
generierter Code
eigener Code
RobotControllerView
MainView
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen


XAML2CPP
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
class MainView : public TMainView<MainView>
{
private:
HINSTANCE hInstance;
BUTTONCLICKHANDLER buttonClickHandler;
public:
MainView(HINSTANCE instance) : hInstance(instance) {}
int LoadMainView();
void RegisterButtonClickHandler(BUTTONCLICKHANDLER handler);
HRESULT btnStartDpws_Click(IXRDependencyObject*, XRMouseButtonEventArgs* args);
...
};
XAML2CPPBase
XAML2CPPPage
TMainView
generierter Code
eigener Code
RobotControllerView
MainView
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen


M – V native – VM
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
View
native
managed
ViewModel
Model
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen

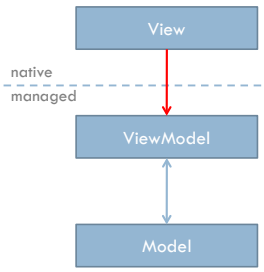

M – V native – VM
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
p/invoke
ROBOTCONTROLLERVIEW_API void OneWayValueChanged(DWORD color)
{
mainView->OneWayValueChanged(color);
}
void MainView::OneWayValueChanged(DWORD color)
{
IXRSolidColorBrushPtr solidBrush;
app->CreateObject(IID_IXRSolidColorBrush, &solidBrush);
solidBrush->SetColor(color);
elpOneWay->SetFill(solidBrush);
}
[DllImport("RobotControllerView.dll")]
static extern void OneWayValueChanged(UInt32 value);
public UInt32 OneWayValue
{
set
{
_oneWayValue = value;
NativeMethods.OneWayValueChanged(_oneWayValue);
}
}
View
native
managed
ViewModel
Model
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen


M – V native – VM
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
Native callback: 1. register
typedef void (CALLBACK *BUTTONCLICKHANDLER)(LPARAM lParam);
ROBOTCONTROLLERVIEW_API void
RegisterButtonClickHandler(BUTTONCLICKHANDLER handler);
delegate void ButtonClickDelegate(IntPtr lParam);
[DllImport("RobotControllerView.dll")]
static extern void RegisterButtonClickHandler(
ButtonClickDelegate delegate);
View
native
managed
ViewModel
Model
Index
DPWS Implementation
Silverlight for embedded
DPWS Demo
Probleme/Fragen

M – V$_{native}$ – VM

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Native callback: 2. fire




```
HRESULT MainView::btnStartDpws_Click(
    IXRDependencyObject *source, XRMouseButtonEventArgs *args)
{
  buttonClickHandler((LPARAM)TEXT("Start"));
  return S_OK;
}
```

```
private void OnButtonClicked(IntPtr lParam)
{
  string button = Marshal.PtrToStringUni(lParam);
  Program.Logger.Debug("Button '" + button + "' clicked!");
}
```

Index | DPWS Implementation | Silverlight for embedded | DPWS Demo | Probleme/Fragen

Demo Prototyp

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- DPWS
  - WS-Discovery (Hello, Bye, Probe)
  - Metadata Exchange
  - OneWay Service
  - TwoWay Service
  - Eventing
- Silverlight for embedded
  - MVVM mit nativer View
  - Design in Blend
  - Storyboards

Index | DPWS Implementation | Silverlight for embedded | DPWS Demo | Probleme/Fragen

Probleme/Fragen

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- Kein Multicast bei Emulator
- TCP Pakete werden beim Debuggen mit Phone per USB nicht verschickt.
- Integration von Direct3D
- WS-Discovery vs. DPWS 1.1 Spezifikation
  - urn:docs-oasis-open-**org**:ws-dd:ns:discovery:2009:01
  - urn:docs-oasis-open:ws-dd:ns:discovery:2009:01
- C++ String Typen, welcher verwenden?

Index | DPWS Implementation | Silverlight for embedded | DPWS Demo | Probleme/Fragen

3 Präsentation des Beta Releases vom 26.05.2010

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Robot Control

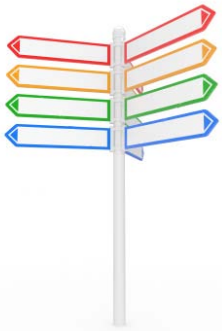
Stefan Züger
Tobias Zürcher

Präsentation der Alpha Version am 26.05.2010

Inhaltsverzeichnis

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- Aktueller Stand
- Demonstration der Anwendung
- Ausblick
- Fragerunde



Index | Aktueller Stand | Demo der Applikation | Ausblick | Fragerunde

Aktueller Stand

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- DPWS
  - Discovery
  - Messaging
  - Description
  - Eventing

Index | Aktueller Stand | Demo der Applikation | Ausblick | Fragerunde

Aktueller Stand

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- Robot Controller
  - Steuern des Roboter (Display & Tastatur)
  - Notaus (& Konfiguration Auto-Notaus)
  - Robo-LAN
    - Notaus / Notaus auflösen
  - Diverse SL Animationen auf Menü/Buttons
  - Alarme (Konfiguration, Auslösung)
  - Betriebsdaten anzeigen
  - Treiberfehler anzeigen

Index | Aktueller Stand | Demo der Applikation | Ausblick | Fragerunde

Aktueller Stand

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

- Remote Controller
  - Device Discovery
  - Session Handling
    - Nur eine Verbindung, auto reconnect
  - Steuern des Roboter
  - Notaus auslösen
  - Betriebsdaten anzeigen
  - Auto-Notaus
  - Alarm

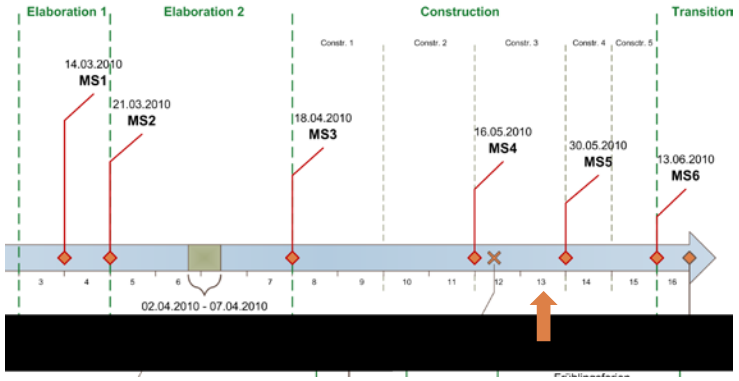
Index | Aktueller Stand | Demo der Applikation | Ausblick | Fragerunde

Demo

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Index | Aktueller Stand | Demo der Applikation | Ausblick | Fragerunde

Ausblick



Ausblick

- Kritische Features
  - DPWS Security & WS-Policy (auch nicht im Micro Framework)
  - „Roboter Controller agiert als Remote Controller“
  - Visual Studio Integration → simple Lösung

Fragerunde

Anhang

Version 1.0

Bachelorarbeit
Roboter Controller Showcase

Projektmitglieder
Stefan Züger
Tobias Zürcher

Betreuer
Prof. Hansjörg Huser

Industriepartner
Zühlke Engineering AG, Schlieren

Inhaltsverzeichnis

1 Glossar

Begriff	Beschreibung
Alarm	Ein Alarm wird ausgelöst, wenn eine bestimmte Alarmgrenze, welche vom Benutzer gesetzt wurde, überschritten wird. Diese Alarmgrenzen beziehen sich in dieser Arbeit ausschliesslich auf die Betriebszeiten der einzelnen Motoren des Roboters.
Betriebsdaten	Betriebszeiten der einzelnen Motoren des Roboters
Client	Gegenstück vom Device: Konsumiert die vom Device angebotenen Services.
Delegate	Typensicherer Funktionspointer im .NET Framework
Device	Gerät, welches Services mithilfe von DPWS zur Verfügung stellt.
DPWS	Device Profile for Web Services: spezifizierter Standard mit dem es ermöglicht werden soll, Web Services auf embedded Systemen einzusetzen. *[OASIS]*
Fehlerzustand	Der Roboter Treiber kann pro Achse/Motor einen generellen Fehler werfen, welcher eine Fehlernummer (1 Byte) enthält. Ein Fehler vom Treiber wird lediglich auf dem Display dargestellt und sonst nicht besonders behandelt.
HH	Hansjörg Huser
JJ	Jürg Jucker
Kunde	In diesem Projekt ist Zühlke Engineering AG der Kunde/Auftraggeber.
M1 – M5	Achsennamen des Playtastic NC-1425 Roboters
Managed code	Code geschrieben in einer .NET Sprache (C#, VB.NET…), Code wird „gemanaged“ von der .NET Runtime.
MKL	Mario Klessascheck
MKO	Michael Koster
Native code	Code geschrieben in C oder C++, der direkt für das Zielsystem kompiliert werden muss
Notaus	Der Notaus Zustand bedeutet der sofortige Stopp der ganzen Robotersteuerung auf dem aktuellen Roboter Controller. Ein Notaus kann manuell oder automatisch ausgelöst werden.
P/Invoke	Platform Invocation Services Funktionalität der Microsoft Common Language Runtime um aus managed Code nativen Code aufzurufen
Pairing	Pairing bedeutet der Zusammenschluss zwischen Remote Controller und Roboter Controller. Beide Parteien merken sich den Kommunikationspartner und können keine weitere Verbindungen mehr eingehen.
RUP	Rational Unified Process: Vorgehensmodell der Firma Rational Software und benutzt Unified Modeling Language als Notationssprache.

SOAP	Ursprüngliche Abkürzung für Simple Object Access Protocol. Seit SOAP 1.2 ist es kein Akronym mehr, sondern ein eigenständiger Begriff, der als W3C Empfehlung anerkannt wurde. SOAP spezifiziert ein Format zur Übertragung von Nachrichten über Netzwerke.
SZ	Stefan Züger
Teaching	Aufzeichnung von Roboter-Bewegungsabläufen, welche gespeichert und zu einem späteren Zeitpunkt beliebig oft ausgeführt werden können.
TFS	Team Foundation Server 2010, Tool für Versions-, Dokument-, Task- und Buildmanagement.
TZ	Tobias Zürcher
UDP	User Datagram Protokoll
VS	Visual Studio
WCF	Windows Communication Foundation
WS-*	Sammlung mehrere Web Services Standards
WS-Discovery	Web Services Dynamic Discovery Technische Spezifikation, die ein Multicast Discovery Protokoll beschreibt, das gebraucht wird um Dienste in einem lokalen Netzwerk zu finden
WSDL	Web Service Definition Language
Zühlke	Kurzform für Zühlke Engineering AG

Tabelle 1: Glossar

2 Silverlight for Embedded Error Codes

Konstante	Wert
XR_E_BASE	-2142830592
XR_E_PARSER_BASE	-2142830492
XR_E_TYPE_BASE	-2142830392
XR_E_NOT_INITIALIZED	-2142830591
XR_E_INVALID_THREAD_ACCESS	-2142830590
XR_E_INVALID_STATE	-2142830589
XR_E_ELEMENT_NOT_FOUND	-2142830588
XR_E_DUPLICATE_REGISTRATION	-2142830587
XR_E_INVALID_OBJECT	-2142830586
XR_E_INVALID_PROPERTY	-2142830585
XR_E_INVALID_ROOT_FOR_CREATING_HOST	-2142830584
XR_E_ABSTRACT_BASE_CLASS	-2142830583
XR_E_PARSER_MISSING_DEFAULT_NAMESPACE	-2142830491

XR_E_PARSER_UNKNOWN_ELEMENT	-2142830490
XR_E_PARSER_UNKNOWN_ATTRIBUTE	-2142830489
XR_E_PARSER_UNKNOWN_NAMESPACE	-2142830488
XR_E_PARSER_INVALID_PROPERTY	-2142830487
XR_E_PARSER_MULTIPLE_PROPERTY_ELEMENT_VALUES	-2142830486
XR_E_PARSER_INVALID_ATTRIBUTE	-2142830485
XR_E_PARSER_INVALID_ATTRIBUTE_VALUE	-2142830484
XR_E_PARSER_ATTRIBUTE_OUT_OF_RANGE	-2142830483
XR_E_PARSER_ATTRIBUTE_READONLY	-2142830482
XR_E_PARSER_FAILED_RESOURCE_FIND	-2142830481
XR_E_PARSER_RESOURCE_KEY_AND_NAME_SET	-2142830480
XR_E_PARSER_TEXT_CONTENT_UNSUPPORTED	-2142830479
XR_E_PARSER_INVALID_CONTENT	-2142830478
XR_E_COLLECTION_DUPLICATE_NAME	-2142830382
XR_E_COLLECTION_ELEMENT_ALREADY_ASSOCIATED	-2142830381
XR_E_ELEMENT_NOT_CREATED	-2142830380
XR_MAXIMUM_CUSTOM_CONTROL_NAME_LENGTH	64
XR_MAXIMUM_NAMESPACE_LENGTH	256
XR_E_STORYBOARD_BEGIN_INVALID_PROPERTY	-2142830391
XR_E_STORYBOARD_BEGIN_INVALID_TARGET	-2142830390
XR_E_STORYBOARD_BEGIN_NO_TARGET	-2142830389
XR_E_STORYBOARD_BEGIN_INCOMPATIBLE_TYPE	-2142830388
XR_E_STORYBOARD_BEGIN_ANIMATION_COMPOSITION	-2142830387
XR_E_STORYBOARD_BEGIN_INVALID_KEYTIME	-2142830386
XR_E_STORYBOARD_MUST_BE_ROOT	-2142830385
XR_E_STORYBOARD_SKIPTOFILL_NO_DURATION	-2142830384
XR_E_STORYBOARD_MODIFY_ACTIVE_ANIMATION	-2142830383
XR_E_COLLECTION_DUPLICATE_NAME	-2142830382
XR_E_COLLECTION_ELEMENT_ALREADY_ASSOCIATED	-2142830381
XR_E_ELEMENT_NOT_CREATED	-2142830380

Tabelle 2: Silverlight for Embedded Error Codes

3 Literaturverzeichnis

[Gamma95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. 1995 *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc.

[Kühner08] Kühner, J 2008 Expert .NET Micro Framework, Apress
http://www.apress.com/book/view/159059973x

[OASIS] http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html
März, 2010

[Url2.1] Apache 2.0 Lizenz
http://www.apache.org/licenses/LICENSE-2.0.html

[Url2.2] WSDAPI
http://msdn.microsoft.com/en-us/library/aa826001(VS.85).aspx

[Url2.3] DPWS im :NET Micro Framework
http://msdn.microsoft.com/en-us/library/ee435393.aspx

[Url2.4] WS4D.org Java Multi Edition DPWS-Stack
http://sourceforge.net/projects/ws4d-javame/

[Url2.5] WS4D-gSOAP
http://trac.e-technik.uni-rostock.de/projects/ws4d-gsoap

[Url2.6] Ermittlung von CommandBars für Visual Studio Add-in Entwicklung
http://blogs.msdn.com/b/dr._ex/archive/2007/04/17/using-ivsproffercommands-to-retrieve-a-visual-studio-commandbar.aspx

[Url2.7] .NET Micro Framework Porting Kit
http://www.microsoft.com/downloads/details.aspx?FamilyID=16fa5d31-a583-4c0d-af74-f4d5e235d5bc

[Url2.8] XAML2CPP Tool von Valter Minute
http://geekswithblogs.net/WindowsEmbeddedCookbook/category/10948.aspx

[Url2.9] MVVM Pattern
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[Url2.10] Silverlight for Embedded Error Codes
http://msdn.microsoft.com/en-us/library/ee501497.aspx

[Url7.1] Offizielle Apache log4net Webpräsenz
http://logging.apache.org/log4net/

Alle Weblinks sind im Juni 2010 auf Ihre Erreichbarkeit überprüft worden.

4 Erklärung über die eigenständige Arbeit

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.

Ort, Datum: Stefan Züger

_____ _____

Ort, Datum: Tobias Zürcher

_____ _____