Seminar/Projekt

Proof-Of-Concept: Webservice-Hosting auf eingebetteten Systemen

Studienrichtung Telematik
Fakultät für Informatik
Sommersemester 2010

Genewein Tim, BSc

Betreuer: Univ.-Prof. Dipl.-Ing. Dr.techn. Wotawa Franz
Institut: Institut für Softwaretechnologie
Datum: 31. Juli 2010

Firma: NTE Systems
Betreuer: Dipl.-Ing. Hafellner Andreas

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 31. Juli 2010 _____

(Unterschrift)

Vorwort

Die vorliegende Arbeit wurde im Sommersemester 2010 im Rahmen eines Seminar-Projektes erstellt. Die Wahl des Themas soll einerseits einen Kontrast zu den Schwerpunkten, die ich im Masterstudium gesetzt habe, bilden - andererseits bin ich persönlich an der Softwareentwicklung für eingebettete Systeme seit meinen Anfängen in dieser Domäne während meiner Ausbildung an der HTL interessiert.

Im .NET Micro Framework habe ich eine Technologie gesehen, welche demonstriert wie die **moderne** Softwareentwicklung für eingebettete Systeme aussieht. Die Möglichkeit *Managed Code* (konkret: .NET Code mit Microsofts exzellenter IDE - Visual Studio) auf Plattformen mit stark limitierten Ressourcen zu verwenden, war Hauptbeweggrund für den Entschluss zu dieser Arbeit.

Mein besonderer Dank gilt Herrn Dipl.-Ing. Stasny Georg, der mir die Mitarbeit an diesem Projekt ermöglicht hat. Persönlich besonders wertvoll waren für mich die Integration in das Entwicklungsteam, das Kennenlernen und Festigen der Abläufe aus dem Berufsalltag und nicht zuletzt das aktive Einbringen von persönlichen Ideen und Meinungen. Gerade diese Fähigkeiten sind schwer zu vermitteln und lassen sich nur durch konsequente Anwendung in entsprechendem Umfeld festigen.

Weiters möchte ich mich bei Herrn Dipl.-Ing. Hafellner Andreas bedanken, der das Projektteam geführt hat und als Ansprechpartner für fachliche, organisatorische aber auch menschliche Fragen immer zur Verfügung stand. Dieser Dank gilt auch dem gesamten Projektteam, welches mir die Mitarbeit am Projekt in vielerlei Hinsicht erleichtert hat.

Von Seiten der Universität wurde ich von Herrn Univ.Prof. Dipl.-Ing. Dr. Franz Wotawa betreut. Bei ihm und dem Institut für Softwaretechnologie möchte mich vor allem für die stets pragmatische und unbürokratische Betreuung bedanken.

Genewein Tim, BSc

Abstract

The main goal of this project thesis is the implementation of a *Proof-Of-Concept for using webservice-communication on embedded devices*. Webservices have a great potential - especially for heterogeneous, distributed systems. However, most embedded platforms still do not offer webservice capabilities. The main reason is an inherent performance-overhead when using this means of communication (which involves a heavy use of XML-parsing).

Microsoft's .NET Micro Framework is a software-platform for running managed-code applications on systems with very limited resources. The Micro Framework also comes with a native support for webservices: the DPWS-stack. In the course of this project, the capabilities of that DPWS-implementation shall be evaluated. Furthermore, possible latencies and data-throughput rates are to be determined. The outcome of these measurements should facilitate further technology- and strategy-decisions.

Another aspect of this project is the evaluation of work flows or *dos and don'ts* in the context of these webservices. The results will be used to help future developers to avoid common pitfalls and shorten their training period.

For the proof-of-concept, the implementation of a data up-/download service has been chosen. It allows the up- and download of files (or binary-arrays) to and from the Micro Framework device. Not only does it involve the transmission and processing of rather large blocks of data; due to its implementation it will also show the transmission-performance for lightweight objects. Based on these two use cases, latency and data-throughput measurements are possible.

At a very early stage in the project, it became clear, that the DPWS-stack implementation is still full of bugs and has not yet reached the required degree of maturity. During the project, several bugs and interoperability-issues could be submitted to and fixed by the Micro Framework Team. At the end of the project, the DPWS-implementation has been improved and with the final release of the *.NET Micro Framework 4.1* all submitted bugs have been fixed.

Due to the parallel development process of the hardware's Ethernet-interface, it was impossible to determine quantitative results during the project. However, it was possible to run the measurements using an emulator and to further achieve some qualitative results (also using the »real hardware«). It turns out, that webservices do require a certain amount of performance and their use should be well designed.

The technology of the Micro Framework and its webservices are ready to be used in commercial development processes - but special care needs to be taken on the decision to use these technologies, since they introduce some limitations on the possible usage scenarios (like the lack of real-time capability, or system response times, etc.). On the other hand, if these limitations can be tolerated, the Micro Framework comes with a vast amount of functionality, that would probably take a few man-years to develop.

Inhaltsverzeichnis

1

1 Einleitung

1.1 Motivation

Mit zunehmender Verfügbarkeit von Rechenleistung im Bereich der eingebetteten Systemen im Niedrig- und Niedrigst-Preissegment, stieg in den letzten Jahrzehnten die Komplexität der Software für derartige Systeme beständig. Vor allem im letzten Jahrzehnt wurden heterogene, verteilte Systeme aufgrund ihrer Modularität und gleichzeitigen Fähigkeit über eine große Bandbreite sehr gut zu skalieren, immer häufiger eingesetzt. Die klaren Grenzen zwischen hochspezialisierten eingebetteten Systemen und klassischen *General-Purpose*-Systemen verschwinden zunehmend. Trotzdem bestehen nach wie vor mitunter große *Unterschiede* zwischen der Entwicklung im »High-Level-« und »Embedded-« Bereich. Obwohl beide Domänen immer mehr zusammenwachsen und diverse Paradigmen aus der klassischen Softwareentwicklung längst Einzug in den Embedded-Bereich gefunden haben, sind die verwendeten Tools und Methoden in vielen Bereichen unterschiedlich.

So sind beispielsweise Themen wie Managed Code, Codegenerierung aus Modellen, Design-by-Contract, etc. bei der Entwicklung von eingebetteter Software wenig verbreitet. Immer wieder kommt das Argument eines Performance-Overheads zu tragen. Dass hier aber noch ganz andere Faktoren wie zum Beispiel Entwicklungszeit, Wartbarkeit, Modularität oder Interoperabilität einen wesentlichen Einfluss haben, wird meist nicht argumentiert. Aus technologischer Sicht, wären viele dieser Konzepte mittlerweile auch auf ressourcenschwachen Systemen sinnvoll ein- und umsetzbar. Trotzdem sind in der Praxis monolithische oder schichten-basierte Architekturen auf Basis von C oder C++ Stand der Technik. Umgekehrt wagen sich auch High-Level-Entwickler selten an die Entwicklung von Hardware-nahem Code - dort stehen die bekannten Tools und Abläufe oft nicht zur Verfügung, Debugging und Testen gestaltet sich unter Umständen recht unkomfortabel.

Das .NET Micro Framework hat das Potential diese Lücke zumindest zu verkleinern, denn es bietet einerseits High-Level-Entwicklern die Möglichkeit in einer vertrauten Umgebung, mit bekannten Tools und Konzepten zu arbeiten, andererseits finden Embedded-Entwickler auch hier ihre Hardware-Nähe bis zu einem gewissen Grad wieder und kommen trotzdem in Kontakt mit neuen Paradigmen und Tools. Vielleicht kann gerade diese Brückentechnologie Vorurteile auf beiden Seiten abbauen.

1.2 Zielsetzung

Die Hardware-Entwicklungsabteilung der Firma *NTE Systems* (Hardware und zugehörige Firm-/Software) hat sich zur Neuentwicklung eines Gerätes (im Folgenden oft einfach als *Hardware* bezeichnet) mit dem .NET Micro Framework als Basis für die zugehörige Software entschlossen.

Unter anderem, weil das Potential dieser neuen Technologie ausgenutzt werden soll und die angesprochene Lücke zwischen Hard- und Softwareentwicklung langfristig geschlossen werden soll. Es besteht großes Interesse daran, Paradigmen wie zum Beispiel die Modellbasierte Entwicklung in die Entwicklungsprozesse der Abteilung zu assimilieren. Darüber hinaus bietet das Micro Framework eine Vielzahl an Funktionalität, welche erst (mühsam) entwickelt werden müsste; die *Time-To-Market* soll durch den Einsatz drastisch verkürzt werden können.

Entwickelt wird ein Gerät zur Steuerung von elektrischen und elektromechanischen Elementen. Ähnlich einer klassischen SPS soll das Steuergerät programmierbar sein und Daten auch über eine Ethernet-Verbindung oder das Internet zur Verfügung stellen. Da das Micro Framework bereits mit einer Implementierung eines Webservice Stacks ausgeliefert wird, soll evaluiert werden, ob Webservices für die auftretenden Anwendungsfälle im Bereich der Kommunikation eingesetzt werden können.

Neben einer Einarbeitungs- und Recherchephase, steht ein *Proof-Of-Concept (POC)* im Mittelpunkt der Arbeit. Anhand dieses POC soll dann abgeschätzt werden, welche Performance mit der Kommunikation über Webservices erreichbar ist und ob damit alle Anforderungen an die Kommunikationsschnittstelle erfüllt werden können. Darüber hinaus soll im Rahmen der Implementierung nach möglichst einfachen und effizienten Arbeitsabläufen gesucht werden - es gilt den Weg für die Produkt-reife Implementierung *zu ebenen* und eventuelle Unsicherheiten oder Schwierigkeiten zu finden und gegebenenfalls zu beheben.

Im Zuge des Projektes soll keine vollständige Implementierung aller Funktionen des späteren Kommunikationsmoduls durchgeführt werden. Auch sollen im Rahmen des Projektes möglichst alle Unklarheiten und Schwierigkeiten aufgezeigt werden - erst an zweiter Stelle steht deren Behebung. Die Ergebnisse des Projektes sollen für spätere Strategie- bzw. Technologie-Entscheidungen und -Abschätzungen herangezogen werden können und die Einarbeitungsphase der produktiven Entwicklung erheblich verkürzen.

1.3 Aufbau der Arbeit

In Kapitel 2 werden zunächst die nötigen Grundlagen (SOA) und Technologien (Webservices, DPWS, WCF) kurz erläutert. In Kapitel 3 wird dann auf die konkrete Realisierung im Rahmen des Projektes eingegangen. Zunächst werden dort Ziele und relevante Kriterien vorgestellt, dann die Umsetzung um diese zu erreichen. Kapitel 4 präsentiert die gewonnen Ergebnisse und gliedert sich in messbare und nicht-quantifizierbare Resultate. Im Kapitel 5 werden die Ergebnisse interpretiert und mögliche Konsequenzen diskutiert. Zuletzt folgt die Zusammenfassung und ein Ausblick in Kapitel 6.

2 Grundlagen

2.1 .NET Micro Framework

Das .NET Micro Framework (NetMF oder MF) ist Microsofts kleinste Plattform für Managed Code. Laut [8] läuft das MF bereits auf 32-Bit Prozessoren, die mit 64kB RAM und 256kB ROM ausgestattet sind und nicht über eine MMU verfügen müssen. Wenn der volle Feature-Umfang, wie zum Beispiel GUI- oder Webservice-Bibliotheken, genutzt wird, sollte die Hardware-Plattform über etwas mehr Speicher verfügen (typische MF-Devices haben 8MB RAM und 4MB Flash-ROM). Lauffähige Versionen gibt es vor allem für ARM7 und ARM9 Derivate, allerdings folgen laufend neue Portierungen wie z.B, auf den Cortex M3.

Das Micro Framework bietet eine Laufzeitumgebung für die Interpretation (!) von .NET - Code, wobei momentan nur C# und ein Subset von .NET (das .NET Micro Framework) unterstützt werden. In großen Teilen entspricht dieses aber, zumindest von den Namenskonventionen her, dem *großen* .NET Framework, sodass der selbe Code auf beiden Plattformen verwendbar ist. Durch die Interpretation des .NET Codes kann das Micro Framework von der Performance her nicht mit nativen Implementierungen auf der selben Hardware mithalten. Von einer Echtzeit-Fähigkeit ist allein aufgrund des Garbage-Collectors nicht zu sprechen.

Prinzipiell ist das Framework ohne weiteres Betriebssystem lauffähig - wenn es die Anwendung erfordert, kann es allerdings auch z.B. auf einem Echtzeit-fähigen Betriebssystem laufen.

Die Vorteile des Frameworks liegen vor allem in der geringeren Entwicklungszeit und seinem Funktionsumfang. Für die portierten Prozessoren wurden große Teile der Hardware gekapselt und können bequem über C#-Objekte und -Methodenaufrufe angesprochen und verwendet werden. Die Anzahl der Features die dadurch mit-ausgeliefert wird ist beeindruckend; unter anderem:

- RS232-, SPI- und I²C-Kommunikation
- Ethernet-Kommunikation inklusive HTTP und HTTPS-Sockets
- Ver- und Entschlüsselung
- Ansteuerung von grafischen Displays und Touchscreens
- Umfangreiche Bibliothek von GUI-Elementen
- Ansteuern von analogen und digitalen Ein-/Ausgängen der Hardware (inkl. PWM)
- Timer
- Multithreading
- XML-Parser und (De-)Serialisierer
- Webservice-Stack (DPWS) inklusive zugehörigem Codegenerator-Tool

- Simples Filesystem
- Möglichkeit über RS232, USB oder Ethernet die Hardware anzubinden
- Debugging auf der realen Hardware von Visual Studio aus.

Durch diese Menge an Features, die »Out-Of-The-Box« verfügbar sind, kann die Entwicklungszeit für einen Prototypen oder auch ein marktreifes Produkt enorm verkürzt werden. Darüber hinaus bieten einige spezialisierte Hersteller passende Hardware mit zugehörigen Treibern für das Micro Framework; wodurch Prototypen oder Produkte, deren Entwicklung Monate in Anspruch genommen hätte, innerhalb von wenigen Wochen umgesetzt werden können (siehe Fallstudie unter: [2]). Nochmals soll hier explizit auf die einfache und komfortable Verwendung der beschriebenen Features hingewiesen werden (C#-Code, ohne sich großartig um bestimmte Register oder Funktionszeiger kümmern zu müssen).

Ein Beispiel: Fast alle Hersteller bieten API-Funktionen für ihre SPI-Implementierungen an - wer die Praxis kennt, weiß aber, dass üblicherweise ein gewisser Entwicklungsaufwand nötig ist, um die Schnittstelle wirklich in einen funktionalen Zustand zu bringen. Meist müssen zur Initialisierung bestimmte Register versorgt werden; zur Kommunikation kommen zusätzlich noch Call-back-Funktionen oder Interrupt-Service-Routinen zum Einsatz. Alles bekannte Konzepte, die sich mit mäßigem Aufwand verwenden lassen - ein Einlesen in die Dokumentation des SPI-Moduls ist aber nach wie vor nötig. Im Falle des MF wurde aber bereits ein derart hoher Grad an Abstraktion geschaffen, dass im Prinzip die Schnittstelle nur instantiiert werden muss und dann mit Methodenaufrufen und Event-Handlern eine Kommunikation eingerichtet werden kann. Das Bild vom »Auspacken und Loslegen« wird hier in bisher unbekannter Weise erreicht.

Details wie das angesprochene Beispiel oder die exzellente Entwicklungsumgebung mit voller Debugging-Funktionalität tragen dazu bei insgesamt die Produktivität jedes Entwicklers zu erhöhen und gleichzeitig die Qualität des Codes zu verbessern.

Das Framework selbst gliedert sich in mehrere Schichten: von einem Treiber- und Hardware-Abstraktions-Layer über die Common-Language-Runtime(CLR), welche für die Interpretation und Ausführung des Managed Codes zuständig ist, bis hin zu Treibern und Bibliotheken in Managed-Code. Seit Herbst 2009 ist auch der komplette Sourcecode des Frameworks (auch die nativen Teile) im sogenannten Porting-Kit frei verfügbar. Dadurch ist es möglich das Framework auf einen anderen Prozessor zu portieren oder zum Beispiel um eigene Treiber zu erweitern. Im nativen Teil des Frameworks wird in C++ entwickelt, für das Erzeugen und Testen des Frameworks steht dort leider nicht Visual Studio zur Verfügung (da jede Zielhardware ihre eigenen Compiler und Tools benötigt, kommt MSBuild zum Einsatz).

Umfassende Informationen, die auch als Basis dieser Arbeit dienen, sind in [13] und [16] zu finden.

2.2 SOA

2.2.1 Allgemein

SOA steht für **serviceorientierte Architektur** (*engl.: service-oriented architecture*) und ist ein Begriff der vor allem im letzten Jahrzehnt oft als Schlagwort in diversen Marketing-Abteilungen

hergehalten hat. SOA lässt sich nicht scharf definieren - gerade deswegen wurden Umsetzungen oft als serviceorientiert verkauft.

Im Wesentlichen bedeutet SOA das Herunterbrechen eines Geschäftsprozesses in funktionale Einheiten, wobei ein Service genau einer solchen Einheit entspricht. Durch Hintereinanderausführen von mehreren Services kann schließlich der gesamte Geschäftsprozess abgebildet werden. Services sollen möglichst unabhängig von anderen Services sein, um deren Wiederverwendbarkeit und Modularität zu steigern («*Loose Coupling*«). Zur Definition eines Services gehört lediglich dessen Schnittstelle (*engl.: interface*), die zugehörige Logik wird versteckt sodass die Semantik eines Services oft unklar ist und separat dokumentiert werden muss. Andererseits wird somit zumindest teilweise die Interoperabilität zwischen heterogenen Teilnehmern gewährleistet. Daher ist die SOA besonders beim Design von verteilten, heterogenen Systemen sehr beliebt. Näheres zur SOA ist in [12] zu finden.

2.2.2 Webservices

Webservices sind eine konkrete Ausprägungsform der Implementierung von SOA. Man spricht von sogenannten *Service Hosts*, d.h. Geräten die einen bestimmten Service anbieten und *Clients* die den Service konsumieren. Natürlich kann ein Host auch mehrere Services gleichzeitig anbieten.

Die Spezifikationen der Webservices sind äußerst umfangreich und nicht immer scharf, teilweise sogar widersprüchlich (in verschiedenen Versionen), sodass die konkreten Webservice-Implementierungen untereinander oft nicht vollständig kompatibel sind und die Interoperabilität verschiedener Implementierungen nicht gewährleistet werden kann. Dies hängt zum Teil auch damit zusammen, dass hinter den Webservice-Spezifikationen große Organisationen (W3C, OASIS, ...) stecken, welche den Fortschritt der Technologie mit großen Schritten voran drängen, um nicht selbst von Konkurrenz-Technologien verdrängt zu werden.

In den sogenannten *WS-** Definitionen, werden die Standards für den Austausch von Nachrichten, Adressierung, das Versenden von Events oder Fehlerbenachrichtigungen, Authentifizierung, Verschlüsselung, etc. festgelegt. Jede dieser Definitionen liegt unter Umständen in verschiedenen Versionen vor. Die Zusammenfassung von einer Teilmenge der Spezifikationen in ganz bestimmten Versionen wird *Profil* genannt.

Folgende Technologien sind unmittelbar mit Webservices verknüpft:

XML (Extensible Markup Language):
: Maschinenlesbares Format zur Kodierung von Daten. XML ist eine universelle Markup-Language, da sie im Prinzip beliebig erweiterbar ist. Sowohl die Beschreibung von Webservices als auch der Datenaustausch basieren auf XML. Dies ist unter anderem einer der großen Kritikpunkte an Webservices im Zusammenhang mit eingebetteten Systemen: Das Übertragen von Daten in XML-Dokumenten bietet zwar Vorteile wie die Lesbarkeit, Validierbarkeit oder die Beschreibung von komplexen Typen, allerdings ist XML auch immer mit einem gewissen Overhead verbunden. Er entsteht einerseits durch die Beschreibung in Text, andererseits durch das Generieren und Parsen von XML-Dokumenten, wobei vor allem letzteres deutliche Performance-Einbußen mit sich bringen kann.

WSDL (Webservice Description Language):
XML-Beschreibung der Struktur von gültigen Nachrichten eines Webservices. WSDL beschreibt sowohl gültige Typen (in Form von XML-Schemata) als auch zulässige Operationen. Leider beschreibt WSDL keine semantischen Informationen zur Verwendung der Operationen.

Es ist üblich auf einem Service-Host unter einer bekannten Adresse die WSDL-Datei zugänglich zu machen. Darüber hinaus ist es üblich den nötigen Code für die Implementierung des zugehörigen Clients (manchmal auch des Services) aus der WSDL-Datei automatisch generieren zu lassen. So gibt es Codegeneratoren für alle bekannten Webservice-Implementierungen wie WCF oder DPWS oder diverse Java-Pendants.

SOAP (Simple Object Access Protocol):
SOAP beschreibt, wie Webservice-Nachrichten in ein gültiges XML-Dokument verpackt werden müssen. Jede SOAP-Nachricht besteht aus einem Envelope, welcher Header (optional) und Body enthält. Hauptsächliche Anwendung von SOAP-Messages sind sogenannte *Remote Procedure Calls* (RPC).

Da jede SOAP-Message ein XML-Dokument ist, darf sie keine in diesem Kontext ungültigen Zeichen enthalten. Sollen beispielsweise binäre Daten in eine SOAP-Message verpackt werden, so werden sie üblicherweise zuerst **Base64**-codiert. Bei dieser Codierung werden aus 8-Bit Binärdaten, 7-Bit ASCII Zeichen. Aus drei Bytes werden dabei vier Base64 codierte Zeichen (ebenfalls Bytes). Dadurch steigt die Größe von binären Daten automatisch um ein Drittel (siehe [1]). Andere Datentypen, wie z.B. Fließkommazahlen werden zur Übertragung in einen String konvertiert. Damit wird deutlich, dass die Übertragung mittels SOAP immer einen nicht ganz unerheblichen Overhead mit sich bringt (sowohl was die Datenmenge als auch die Verarbeitungszeit betrifft).

MTOM (SOAP Message Transmission Optimization Mechanism):
Um große Blöcke binärer Daten zu übertragen empfiehlt die W3C den MTOM-Standard (siehe [10]). Er ermöglicht die Übertragung von Nicht-ASCII-Daten. Im Prinzip besteht eine MTOM-codierte Nachricht aus mehreren *Body-Parts* (sog. Multipart-Message), wobei die ursprüngliche SOAP-Message den ersten Body-Part darstellt. Anstelle der binären Daten enthält sie jedoch eine Referenz auf einen weiteren Body-Part, welcher die binären Daten (uncodiert) trägt. Aus diesem Grund bringt MTOM zunächst einen zusätzlichen Overhead mit sich, der sich im Vergleich zu SOAP erst ab einer gewissen Größe der binären Daten amortisiert (ca. ein Kilobyte). Daher kann MTOM nicht als »Allheilmittel« zur Übertragung eingesetzt werden, sondern stellt vielmehr einen Spezialfall für das Versenden von großen binären Datenblöcken dar.

Ein Webservice setzt sich aus einer Menge von sogenannten Operationen zusammen, wobei eine Operation in vielen Fällen mit einer Funktion/Prozedur zu vergleichen ist. Jede Operation hat einen Eingabe- und einen Ausgabe-Datentyp und wird über einen RPC aufgerufen. Der Input-Datentyp wird dabei in einer *Request-Message* übergeben; der Output-Datentyp folgt in der zugehörigen *Response-Message*. Als Spezialfall gibt es Operationen ohne Rückgabewerte - sogenannte *One-Way-Requests*.

Detaillierte Informationen sind in [12] zu finden.

2.2.3 DPWS

DPWS bedeutet *Devices Profile For Webservices* und ist ein Webservice-Profil mit einem speziellen Fokus auf Ressourcen-schwache Geräte. Das Profil wurde von Microsoft entwickelt und vorgeschlagen und mittlerweile von der OASIS standardisiert (siehe [3]). Aus diesem Grund ist ein DPWS-Stack auf jedem Windows-Betriebssystem seit Windows Vista implementiert. Ebenso wurden die Minimalanforderungen für DPWS auf dem Micro Framework umgesetzt. Großes Manko der Implementierung am MF - die Spezifikationen für WS-Security (Authentifizierung, Verschlüsselung) sind in den DPWS-Spezifikationen nur als optional definiert und fehlen daher am MF.

Mehr Informationen und eine Bewertung der Leistungsfähigkeit von DPWS im Bereich eingebetteter Systeme ist in [12] zu finden.

2.2.4 WCF

Windows Communication Foundation (kurz: WCF) ist Microsofts Implementierung für serviceorientierte Kommunikationstechnologien, wie Webservices, für das .NET Framework. Im Gegensatz zu DPWS wird jedoch nicht nur ein spezielles Profil angeboten - WCF bietet ein sehr umfangreiches Webservice-Framework mit dem sich eine ganze Reihe von Profilen bzw. WS-* Definitionen umsetzen lässt. Darüber hinaus gibt es noch Erweiterungen die (noch) nicht von der OASIS oder W3C standardisiert wurden und daher nur zur Kommunikation zwischen reinen WCF-Teilnehmern verwendet werden können (beispielsweise gibt es ein binäres Binding, welches die Daten »auf der Leitung« binär und somit sehr effizient überträgt - der Standard kennt ein derartiges Binding nicht). Weitere Informationen sind unter [11] zu finden.

Durch seinen Reifegrad und Umfang abstrahiert WCF alle Details der Webservice-Spezifikationen, sodass sich ein Entwickler im Normalfall nicht mit den WS-Standards auseinandersetzen muss. Aufgrund der Komplexität der Spezifikationen erweist sich das als großer Vorteil, solange keine Interoperabilität mit Servern oder Clients anderer Technologien gewährleistet werden muss.

2.2.5 OPC-UA

OPC steht für *OLE for Process Control* und beschreibt eine Technologie zur Standardisierung des Datenaustausches im Bereich der Automatisierungstechnik. In dieser Domäne war es üblich, dass jeder Hersteller ein proprietäres Protokoll zum Austausch von Daten implementiert - darüber hinaus haben Systeme in diesem Bereich durchaus Lebenszeiten von 15 Jahren und mehr. Dadurch hat sich die Technologie in der Domäne relativ träge weiterentwickelt, aber natürlich besteht ein großer Bedarf, die bestehenden Kommunikationskanäle auch über das Internet anbieten zu können. Außerdem sollen Systeme von unterschiedlichen Herstellern »miteinander sprechen« können bzw. mit einem gemeinsamen System zur Visualisierung und Fernwirkung kommunizieren. Als Ergebnis eines langwierigen Standardisierungsprozesses wurde OPC definiert, das allerdings mittlerweile auf veralteten Technologien basiert (Stichwort: DCOM). Die Spezifikationen sind unter [9] einzusehen.

In den letzten Jahren wurde eine Erweiterung des Standards vorgenommen - OPC-UA (*OPC Unified Architecture)*. UA steht für die Umsetzung von SOA im Kontext von OPC. Als Basis

dienen Standardtechnologien und ein frei verfügbarer Stack in C. Dieser Stack bietet die Möglichkeit, Daten binär zu übertragen (wofür allerdings der zugehörige Port freigeschaltet sein muss) oder die Daten über Webservices mit Hilfe von SOAP-Messages auszutauschen.

OPC(-UA) erlaubt es unter anderem einzelne Variablen auszulesen und zu manipulieren, Informationen über alle Variablen und zugehörige Hierarchien zu beziehen, einen Eventing/Alarming-Mechanismus zu nutzen, auf historische Daten zugreifen zu können, etc.

OPC gilt als **der** Standard im Bereich der Automatisierungstechnik und die Tatsache, dass die OPC-Foundation ebenfalls auf SOA und Webservices setzt, deutet in die Richtung, dass diese Technologien für den Einsatz in der Domäne ausgereift und passend sind.

Eine umfassende und detaillierte Beschreibung zur Architektur und Umsetzung von OPC-UA ist in [14] zu finden.

3 Realisierung

3.1 Service-Implementierung

3.1.1 Allgemeine Kriterien

Ziel des Projektes ist ein Proof-Of-Concept, mit dem die Technologie »Webservices auf dem .NET Micro Framework« evaluiert werden kann. Die Ergebnisse dienen später als Basis für Technologie- und Strategie-Entscheidungen. Außerdem sollen im Rahmen des Projektes Stärken, Schwächen und eventuelle Tücken kennengelernt werden, sodass der Weg für eine Produkt-reife Umsetzung geebnet wird. Aus diesem Grund ist die Funktionalität der umgesetzten Services von zweitrangiger Bedeutung. Vielmehr sollen im Projekt möglichst alle relevanten Aspekte für zukünftige Anwendungsfälle abgedeckt oder zumindest Abschätzungen dieser Fälle ermöglicht werden.

Die Entscheidung fiel daher auf die Entwicklung einer Up- und Download-Funktionalität, die im Sinne einer möglichst breiten Abdeckung von Anwendungsfällen mit Zusatzfunktionalität versehen wird, welche nicht immer essentiell nötig ist oder eventuell auch anders umsetzbar wäre.

Entwickelt wurde die Funktionalität Dateien (oder genauer: binäre Arrays von relevanter Größe) auf das MF-Gerät zu übertragen (hochzuladen) und dual dazu wieder herunterzuladen. Für das Szenario der programmierbaren Hardware-Steuerungseinheit hat dies insofern Relevanz, als damit die Übertragung des Steuerungs-Programms oder einer Menge von historischen Datenwerten abgeschätzt werden kann. Als Größenordnung wurde für beide Anwendungsfälle ein Bereich von einigen hundert Kilobyte festgelegt. Da Webservices prinzipiell nicht zur Übertragung von großen Datenmengen konzipiert wurden, herrscht hier die größte Unsicherheit im Bezug auf deren Performance. Sollten die Ergebnisse darauf hindeuten, dass Webservices hier ungeeignet sind, so muss über Alternativen nachgedacht werden.

Konkret sollen Dateien während dem Up- oder Download in Segmente bestimmter Größe zerlegt werden, wobei jedes Segment mit einer eigenen Webservice-Message übertragen wird und an der jeweiligen Gegenstelle wieder zur vollständigen Datei zusammengesetzt werden soll. Da jede Webservice-Nachricht einen Overhead mit sich bringt, entsteht ein Trade-Off zwischen wenigen großen Segmenten und vielen Segmenten, die in jeweils kleineren Messages resultieren, welche sich schneller verarbeiten lassen. Vor allem aufgrund der limitierten Ressourcen der Hardware ist unklar, ob große Segmente bei der Verarbeitung Probleme verursachen (da ja beispielsweise entsprechend große Puffer allokiert werden müssen). Die Größe der soll Segmente frei wählbar sein. Mit der daraus resultierenden Implementierung kann in einer Reihe von Testläufen bestimmt werden, welche Segmentgröße zu einer möglichst optimalen Verarbeitungszeit führt.

Das Festlegen der Segmentgröße soll während der Initialisierung von Up- oder Download zwischen Server und Client verhandelt werden. Um verschiedene Up- und Downloads strikt voneinander zu trennen, erfolgt jede Übertragung im Kontext einer eigenen Session. Hier ist die Rede von leichtgewichtigen Session-Objekten, welche am Server erzeugt werden - Sessions auf Ebene der TCP/IP-Kommunikation (Layer 5 im OSI-Modell) sind in diesem Zusammenhang nicht gemeint. Die Session-Objekte dienen einem weiteren Zweck: Sie entsprechen in etwa simplen Objekten für Variablen, welche vom Steuergerät abgebildet und verwaltet werden (z.B. Variablen für die digitalen Ein- und Ausgänge). Auf Basis von Abschätzungen die mit der Übertragung von Session-Objekten durchgeführt werden, lassen sich somit auch Aussagen für die Übertragung von einfachen Variablen-Typen treffen.

Mit Hilfe der Implementierung sollen vor allem zwei Anwendungsfälle abgedeckt werden:

- Übertragung von großen binären Datenblöcken (mehrere hundert Kilobyte). Entspricht dem Übertragen des Steuerungs-Programms oder einer Anzahl von historischen (archivierten) Variablenwerten. Datenblock soll in Segmenten übertragen werden. Prinzipielle Performance (Datendurchsatz) und Anwendbarkeit dieser Methode, sowie optimale Segmentgröße sollen bestimmt werden.
- Übertragung von relativ kleinen Nutzdaten-Paketen. Entspricht dem Übertagen von einzelnen oder wenigen Variablen-Objekten. Vor allem die Performance im Sinne der Latenzzeiten soll bestimmt werden.

Aus technischen Gründen erfolgt die Implementierung mit **zwei Services** - einem Service zur Übertragung der Datensegmente (unter Verwendung der MTOM-Codierung) und einem Service zur Verwaltung der Übertragung (Verwalten von Sessions und Dateien am Server; Umsetzung in SOAP). Nähere Details sind in den Kapiteln 3.1.2 und 3.1.3 zu finden.

Die Services werden mit Hilfe des Micro Frameworks umgesetzt, wobei ein MF-Device entwickelt wird, welches die angesprochenen Services hosten wird. Als Gegenstelle soll ein WCF-Client zum Einsatz kommen. Die Kommunikation mit dieser Gegenstelle soll vor allem auf Interoperabilitätsprobleme hin untersucht werden.

3.1.2 DataAccessService

Allgemein

Der *DataAccessService* dient zum Austausch der Nutzdaten für Up- und Download. Mit ihm werden einzelne Segmente übertragen; wobei hier die Verwendung einer MTOM-codierten Übertragung angedacht ist. Da jeder Datentransfer nur innerhalb einer (DataAccess)Session stattfinden kann (und dort auch die Segmentgröße für die aktuelle Übertragung festgelegt wird) muss der *DataAccessManagementService* (siehe: Kapitel 3.1.3) zunächst zur Erzeugung der jeweiligen Session verwendet werden.

Der zugehörige Service-Contract ist im Anhang unter: A.2 zu finden. Außerdem liegt der Dokumentation eine interaktive, automatisch generierte HTML-Dokumentation der Services bei.

Das zum Service gehörige Interface (inklusive der darin verwendeten komplexen Datentypen) ist in Abbildung 3.1 dargestellt.

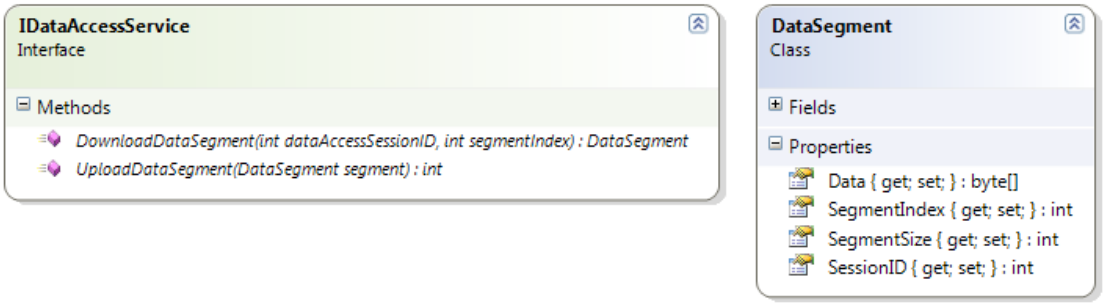

Abb. 3.1: Definition des Service-Interfaces und der darin verwendeten Typen

Typen

Typ: ***DataSegment***

Dieser Typ wird verwendet um ein Segment der Nutzdaten zu verpacken und mit Hilfe der Operationen des DataAccessService zu übertragen. Das zugehörige Klassendiagramm ist in Abbildung 3.1 zu sehen.

Data: Byte-Array, welches die zu übertragenden Nutzdaten enthält.

SegmentIndex: Index des Segmentes; wird für die Zuordnung des Segmentes innerhalb der Datei benötigt.

SegmentSize: Größe des Nutzdaten-Segmentes (Byte-Arrays) in Byte.

SessionID: Identifier der zugehörigen DataAccessSession.

Operationen

DownloadDataSegment

Mit dieser Operation wird ein Segment einer Datei vom Client aus bezogen. Spezifiziert wird das Segment durch die Session (welche sowohl Dateiname als auch Segmentgröße beinhaltet) und den Index des Segmentes innerhalb der gesamten Datei. Rückgabewert der Operation ist ein Objekt vom Typ DataSegment. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.2 dargestellt.

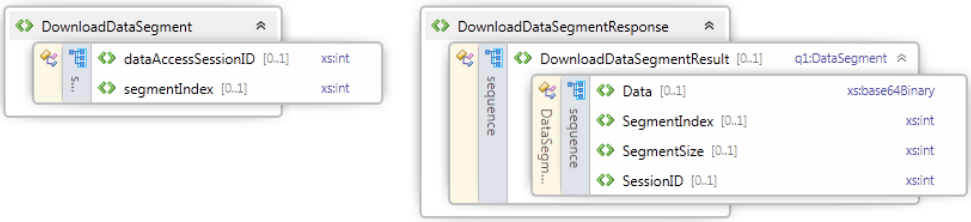

Abb. 3.2: XSD-Schema der Request- und Response-Message der Operation DwonloadDataSegment

UploadDataSegment

Mit dieser Operation wird ein Segment einer Datei vom Client zum Server hochgeladen. Spezifiziert wird das Segment durch ein Objekt vom Typ DataSegment. Im Falle einer erfolgreichen Übertragung liefert die Operation den Wert 0 zurück. Anmerkung: In den meisten Fällen wird bei einer nicht-erfolgreichen Übertragung eine WS-Fault-Message generiert und zurückgegeben; sie wird Client-seitig in eine Exception transformiert. Der Integer-Rückgabewert wurde hier zu Demonstrationszwecken eingeführt. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.3 dargestellt.

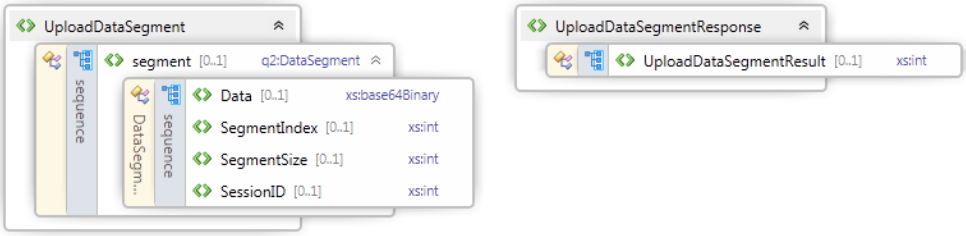

Abb. 3.3: XSD-Schema der Request- und Response-Message der Operation UploadDataSegment

3.1.3 DataAccessManagementService

Allgemein

Der *DataAccessManagementService* dient zur Verwaltung der *DataAccessSessions*, welche vom *DataAccessService* (siehe: 3.1.2) für die Up- und Download-Operationen benötigt werden. Darüber hinaus bietet er Funktionalität an, um Dateien vom Server zu löschen bzw. eine Liste mit allen am Server abgelegten Dateien beziehen zu können.

Der zugehörige Service-Contract ist im Anhang unter: A.2 zu finden. Außerdem liegt der Dokumentation eine interaktive, automatisch generierte HTML-Dokumentation der Services bei.

Das zum Service gehörige Interface (inklusive der darin verwendeten komplexen Datentypen) ist in Abbildung 3.4 dargestellt.

Typen

Typ: ***DataDataAccessSession***

Dieser Typ dient als Basis für alle Up- oder Download-Transaktionen. Er wird vor dem Beginn einer Übertragung instantiiert und hält Informationen über die zu übertragende Datei und die zu verwendende Segmentgröße. Alle weiteren Operationen mit der zugehörigen Datei laufen dann im Kontext dieser Session ab, wobei sowohl Client als auch Server relevante Information aus dem entsprechenden DataAccessSession-Objekt beziehen. Das zugehörige Klassendiagramm ist in Abbildung 3.4 zu sehen.

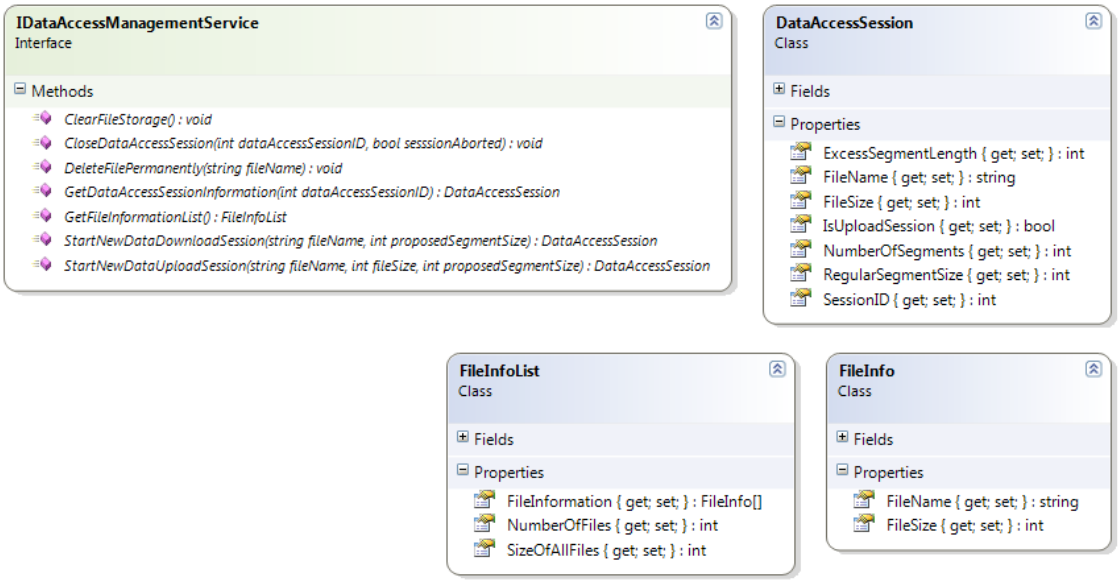

Abb. 3.4: Definition des Service-Interfaces und der darin verwendeten Typen

ExcessSegmentLength: Bei der Zerlegung in Segmente fixer Größe, ergibt sich höchstwahrscheinlich ein Segment mit einer irregulären Größe (der Übertrag; das letzte Segment). In diesem Feld wird die Größe dieses »*überstehenden*« Segmentes in Bytes angegeben.

FileName: Der Dateiname der zu übertragenden Datei. Am Server werden Dateien eindeutig über ihren Namen identifiziert, sodass ein Upload nicht möglich ist, wenn bereits eine Datei mit dem selben Namen existiert. Bei der Anfrage für einen Download werden Dateien genauso über ihren Namen (eindeutig) indentifiziert.

FileSize: Größe der zu übertragenden Datei in Bytes.

IsUploadSession: Boolesches Feld, welches spezifiziert, ob es sich um eine Upload- (oder Download-) Session handelt.

NumberOfSegments: Anzahl der Segmente in welche die Datei zerlegt wird. Wird aufgrund der Dateigröße und der regulären Segmentgröße bestimmt. Hier ist auch ein eventuelles irreguläres Segment mitgezählt.

RegularSegmentSize: Reguläre Segmentgröße - wird beim Initiieren eines Transfers vom Client vorgeschlagen und vom Server daraufhin bestätigt oder abgeändert (wenn der Server Segmente mit der vorgeschlagenen Größe nicht verarbeiten kann).

SessionID: Eindeutiger Identifier der Session - da der Server das Session-Objekt hält, wird in den meisten Folge-Operationen nur mehr über die SessionID gearbeitet - das heißt es kommt nicht zu einer erneuten Übertragung des gesamten Session-Objektes.

Typ: ***FileInfoList***

Dieser Typ wird verwendet um eine Liste aller auf dem Server abgelegten Dateien zu beziehen. Da beim Download über den Dateinamen identifiziert wird, kann diese Liste als

Basis für eine Download-Operation herangezogen werden. Außerdem wird für jede Datei noch ihre Größe mitgegeben. Die beiden Felder *NumberOfFiles* und *SizeOfAllFiles* sind in diesem Fall redundant, da sie beide aus dem Array an *FileInfo*-Objekten bezogen werden könnten. Sie wurden allerdings zur Fehlererkennung eingeführt, da es in den Versionen des Micro Frameworks bis inklusive Beta 4.1 Refresh #3 einen Bug bei der Übertragung von Arrays gab. Das zugehörige Klassendiagramm ist in Abbildung 3.4 zu sehen.

FileInformation: Array aus *FileInfo*-Objekten. Entspricht der Liste aller am Server vorhanden Dateien (d.h. deren Namen und Größe).

NumberOfFiles: Redundantes Feld, welches die Anzahl aller am Server liegenden Dateien angibt.

SizeOfAllFiles: Redundantes Feld, welches die Gesamtgröße (in Bytes) aller am Server liegenden Dateien angibt.

Typ: ***FileInfo***

Simples Objekt, das Information zu einer am Server abgelegten Datei liefert. Das zugehörige Klassendiagramm ist in Abbildung 3.4 zu sehen.

FileName: Name der Datei - eine Datei mit dem selben Namen kann nicht erneut hochgeladen werden.

FileSize: Größe der Datei in Bytes.

Operationen

ClearFileStorage

Mit einem Aufruf dieser Operation werden alle auf dem Server abgelegten Dateien gelöscht. Die Operation wird vorwiegend zur Initialisierung des Systems vor der Ausführung der einzelnen Testfälle verwendet. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.5 dargestellt.

Abb. 3.5: XSD-Schema der Request- und Response-Message der Operation ClearFileStorage

CloseDataAccessSession

Die zu einem Up- oder Download gehörige Session muss nach der vollständigen Übertragung vom Client aus geschlossen werden. Durch den Aufruf dieser Operation wird das zugehörige Session-Objekt am Server zerstört und alle weiteren Operationen, welche die zugehörige SessionID angeben sind ungültig. Eine hochgeladene Datei ist erst nach dem Aufruf dieser Operation (mit dem Flag *sessionAborted* auf false) für den Download verfügbar. Wird das angesprochene Flag auf true gesetzt, werden die bis dahin abgelegten Daten der zugehörigen Session vom Server gelöscht. Für den Download spielt dieses Flag keine

Rolle. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.6 dargestellt.

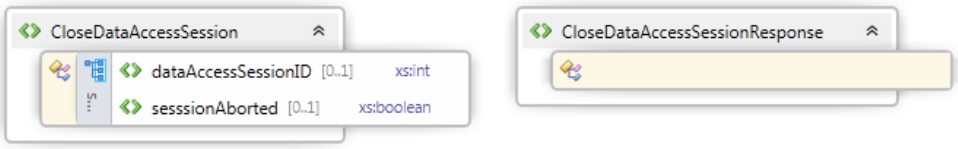

Abb. 3.6: XSD-Schema der Request- und Response-Message der Operation CloseDataAccessSession

DeleteFilePermanently

Mit dieser Operation ist es möglich einzelne Dateien (unwiderruflich) vom Server zu löschen. Zur Spezifikation der Datei muss der (eindeutige) Dateiname angegeben werden. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.7 dargestellt.

Abb. 3.7: XSD-Schema der Request- und Response-Message der Operation DeleteFilePermanently

GetDataAccessSessionInformation

Durch Aufrufen dieser Operation kann bei bekannter SessionID, das zugehörige *DataAccessSesssion*-Objekt vom Server bezogen werden. Das Objekt enthält Informationen zur aktuellen Übertragung, wie z.B. Dateiname und Segmentgröße. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.8 dargestellt.

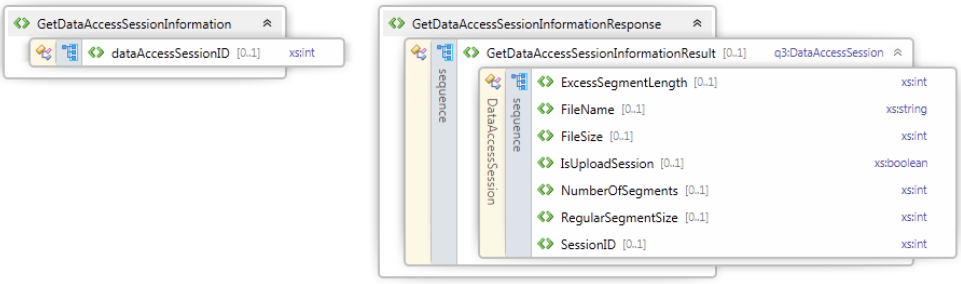

Abb. 3.8: XSD-Schema der Request- und Response-Message der Operation GetDataAccessSessionInformation

GetFileInformationList

Um eine Liste aller auf dem Server befindlichen Dateien zu beziehen, kann die Operation *GetFileInformationList* aufgerufen werden. Sie liefert eine Liste mit allen Dateinamen

und den jeweiligen Dateigrößen zurück. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.9 dargestellt.

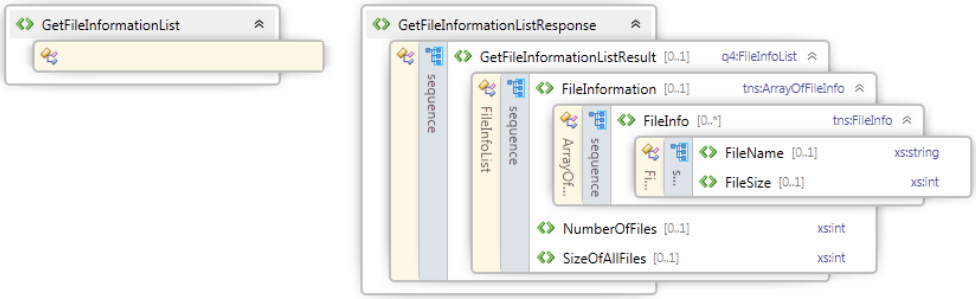

Abb. 3.9: XSD-Schema der Request- und Response-Message der Operation GetFileInformationList

StartNewDataDownloadSession

Ein Aufruf dieser Operation ist vor dem Beginn jedes Downloads nötig. Beim Aufruf gibt der Client den Dateinamen und die vorgeschlagene Segmentgröße mit. Der Server überprüft daraufhin, ob die Datei vorhanden ist und korrigiert die Segmentgröße nach unten falls der Vorschlag das Limit des Servers überschreitet. Anschließend werden vom Server die Anzahl der Segmente und die Größe des letzten Segmentes berechnet. Alle Informationen werden dann in einem neuen *DataAccessSession*-Objekt abgelegt und eine Kopie des Objektes wird an den Client übertragen. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.10 dargestellt.

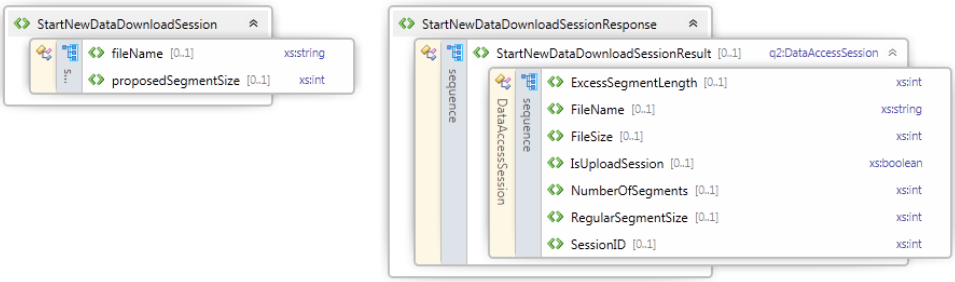

Abb. 3.10: XSD-Schema der Request- und Response-Message der Operation StartNewDataDownloadSession

StartNewDataUploadSession

Ein Aufruf dieser Operation ist vor dem Beginn jedes Uploads nötig. Beim Aufruf gibt der Client den Dateinamen und die vorgeschlagene Segmentgröße, sowie die Gesamtgröße der Datei mit. Der Server überprüft daraufhin, ob die Datei noch nicht vorhanden ist und korrigiert die Segmentgröße nach unten falls der Vorschlag das Limit des Servers überschreitet. Außerdem wird der Speicher für die neue Datei allokiert. Anschließend werden vom Server die Anzahl der Segmente und die Größe des letzten Segmentes berechnet. Alle Informationen werden dann in einem neuen *DataAccessSession*-Objekt abgelegt und eine Kopie des

Objektes wird an den Client übertragen. Die zugehörigen XSD-Schemata für Request- und Response-Typ sind in Abbildung 3.11 dargestellt.

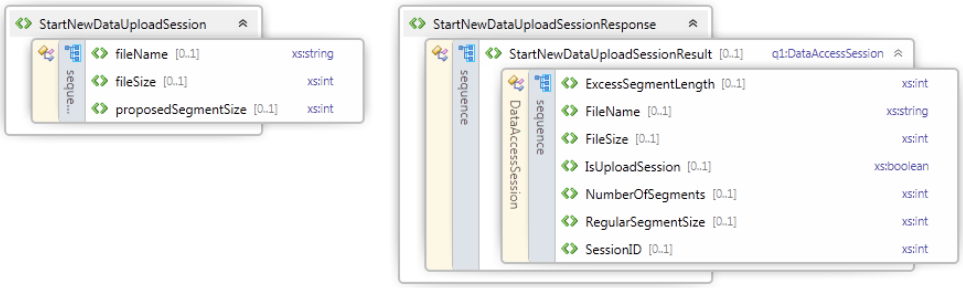

Abb. 3.11: XSD-Schema der Request- und Response-Message der Operation StartNewDataUploadSession

3.2 Tests und Testbed

3.2.1 Allgemeines

Da es sich bei der Implementierung um einen reinen POC handelt, liegt das Hauptaugenmerk beim Aufbau der Testumgebung vor allem auf der Methodik und den einsetzbaren Möglichkeiten. Prinzipiell gilt es auch hier, effiziente und gleichzeitig einfache Möglichkeiten für die Produkt-reife Implementierung aufzuzeigen und zu bewerten.

Die Entwicklung von Managed Code Applikationen für das Micro Framework ist unter Visual Studio mit vollem Debugging-Support möglich. Leider trifft das nicht auf die integrierte Testumgebung zu. In eingeschränkter Form lassen sich zumindest Unit-Tests durchführen. Gerade im Kontext eines Webservices ist es jedoch möglich einen Großteil der Tests Client-seitig auszuführen. Da im Projekt ein WCF-Client zum Einsatz kommt (und dieser mit dem Micro Framework im Speziellen nichts zu tun hat), kann hier die volle Stärke der integrierten Testumgebung von Visual Studio ausgenutzt werden. So lassen sich Codegrundgerüste für Unit-Tests generieren oder es können ganze Testreihen (Ordered Tests, Loadtests, etc.) durchgefahren werden, wobei automatisch die zugehörigen Ergebnisse in eigenen Reports dokumentiert werden. Darüber hinaus kann auch in eingeschränktem Maß die Funktionalität des integrierten Profilers zur Performance-Analyse verwendet werden.

Im Projekt wurden einige Tests als Unit-Tests und Ordered-Tests (Geordnete Ausführung von mehreren Unit-Tests) implementiert. Darüber hinaus gab es die Anforderung bestimmte Performance-Messungen zu implementieren (etwa um die optimale Segmentgröße für Up- und Download bestimmen zu können). Auch diese Messungen wurden als eigene Testfälle umgesetzt, wobei darin noch zusätzliche Ausgaben mit den Ergebnissen der Performance-Messungen generiert werden.

Ausgewählte Testfälle:

Funktionalität / Datenintegrität
: Mit diesem Test (der aus mehreren elementaren Testfällen besteht) wird die korrekte Funktionalität des Services im regulären Betrieb getestet. Im Lauf des Tests wird eine bestimmte Datei auf den Server hochgeladen und anschließend wieder heruntergeladen, wobei zuletzt überprüft wird, ob die Daten der heruntergeladenen Datei exakt denen der Originaldatei entsprechen. Da dieser Test große Teile des Gesamtsystems auf beiden Seiten involviert, kann mit ihm relativ schnell festgestellt werden, wenn einzelne Teile aufgrund von Änderungen nicht mehr funktional sind.

Latenz
: Um die Performance der Übertragung von kleinen Datenpaketen quantifizieren zu können wurde die Messung der Latenz herangezogen. Bei diesen Testläufen werden Up- oder Downloadsessions erzeugt und sofort wieder geschlossen, ohne jemals Nutzdatenpakete zu übertragen (*DataAccessSession*-Objekte werden sehr wohl übertragen). Dabei wird (Clientseitig) jeweils die Zeit vom Absetzen des **ersten** Requests (zur Erzeugung einer neuen Session) bis zum Empfang der **zweiten** Response (vom Schließen der Session) bestimmt - es wird also quasi die doppelte Latenzzeit gemessen. Im Unterschied zum Download, wird beim Upload bereits beim Erzeugen der Session Speicher am Server allokiert. Ziel ist es mit den Ergebnissen dieses Tests abschätzen zu können, welche Latenzzeiten sich beim Übertragen von einzelnen Variablen-Objekten oder Listen von wenigen Variablen-Objekten ergeben. Um stabilere Ergebnisse zu erhalten wird der Vorgang im Lauf des Tests mehrfach wiederholt.

Datendurchsatz bzw. optimale Segmentgröße
: Um die optimale Segmentgröße bestimmen zu können wird eine Datei (ca. 240kB) hoch- bzw. heruntergeladen. Der Vorgang wird mehrfach wiederholt, wobei jeweils eine andere reguläre Segmentgröße verwendet wird. Gemessen wird die Gesamtzeit, die zur Übertragung der ganzen Datei nötig ist. Die Segmentgröße, welche die kürzeste Zeit zur Übertragung in Anspruch nimmt, wird als optimale Segmentgröße (im Kontext der verwendeten Werte) festgelegt. Gleichzeitig, kann aus der Gesamtzeit der Übertragung und der Größe der Datei ein ungefährer Wert für den maximalen Datendurchsatz ermittelt werden. Dieser Testfall dient vor allem als Basis für spätere Strategie- und Technologie-Entscheidungen.

  Als maximale Segmentgröße wurden (vorerst) ca. 42kB festgelegt. Dies ergibt sich aus der Tatsache, dass die maximale Größe für SOAP- (oder MTOM-) Messages **standardmäßig** bei 64kB liegt. Mit den verwendeten 42kB bleibt also neben den Nutzdaten noch Platz für einen gewissen Overhead. Sollte es sich als notwendig erweisen, könnte dieser Wert nahezu beliebig nach oben vergrößert werden - auf Seiten des WCF-Clients durch simples Abändern der Konfiguration; auf Seiten des Micro Frameworks im Prinzip nur durch die Implementierung limitiert. Andererseits ist es vermutlich nicht sinnvoll auf einer Plattform mit derart limitierten Ressourcen Segmente mit mehreren hundert Kilobyte zu verwenden. Darüber hinaus kann unter Einhaltung der Standard-Message-Größe die Interoperabilität mit anderen Clients eher gewährleistet werden.

Um qualitative Aussagen über den Unterschied zwischen der Übertragung der Nutzdaten mit SOAP und MTOM-Codierung treffen zu können, wurden am Micro Framwework Device zwei Versionen des *DataAccessService* (Kapitel 3.1.2) parallel gehostet (einmal die SOAP- und einmal die MTOM-Variante). Leider ist es am Micro Framework, im Gegensatz zu WCF-Implementierungen, nicht möglich zwischen SOAP und MTOM einfach durch einen entsprechenden Eintrag in der

Konfigurationsdatei umzuschalten - es muss für beide Versionen jeweils verschiedener Code generiert werden.

3.2.2 Hardware-Plattform / Emulator

Als Zielplattform kommt das *AUG AMI DevKit* der Firma *AUG Elektronik* (siehe: [4]) zum Einsatz. Kernstück des AMI-Boards ist ein mit 200Mhz getaktetes **Atmel ARM9**-Derivat. Darüber hinaus stehen auf dem Board 64MB SDRAM und 256MB NAND-FLASH Speicher zur Verfügung. Parallel zum Projekt findet auch die Entwicklung einer Ethernet-Anbindung des Boards statt - daher können Webservices auf der realen Hardware erst relativ spät im Projektverlauf getestet werden.

Mit dem .NET Micro Framework SDK wird jedoch ein Emulator mit-ausgeliefert. Im Prinzip ist dieser Emulator eine Portierung des Micro-Frameworks für Windows-Betriebssysteme und entspricht somit in den oberen Softwareschichten sehr genau der realen Hardware. Im nativen Treiber-Layer unterscheiden sich die Implementierungen allerdings. Da auch der Quellcode des gesamten Frameworks verfügbar ist, können der Emulator und das darauf laufende Micro Framework nach Belieben angepasst werden - ein anschließendes Neukompilieren reicht aus, um die Änderungen in den Emulator zu übernehmen. Darüber hinaus ist der Emulator direkt in Visual Studio integriert, sodass Projekte direkt von dort aus gestartet und mit Hilfe der umfangreichen Debug-Funktionalität analysiert werden können. Genauso kann auch die reale Hardware (über RS232, USB oder Ethernet) angebunden und direkt von Visual Studio aus programmiert werden (volle Debugging-Funktionalität steht ebenfalls zur Verfügung).

4 Ergebnisse

4.1 Einführende Bemerkungen

Da der Emulator einer »für Windows kompilierten« Version des Micro Frameworks entspricht, sind mit den daraus gewonnenen Ergebnissen durchaus qualitative Aussagen möglich. Wie sich im Detail zeigt, waren die getroffenen Annahmen richtig und vor allem die Umsetzung mit MTOM hat einen deutlichen Performance-Zuwachs im Fall des Uploads mit sich gebracht. Da jedoch dem Emulator wesentlich mehr Ressourcen als der realen Hardware zur Verfügung stehen (Hardware hat 200Mhz, 64MB RAM), können keine quantitativen Aussagen getroffen werden.

Die Durchführung der Tests auf der realen Hardware ist somit unerlässlich. Leider ist die Entwicklung der Ethernet-Schnittstelle zum momentanen Zeitpunkt noch nicht soweit abgeschlossen, dass von einer Produkt-reifen Qualität ausgegangen werden kann. Dementsprechend ergeben sich bei den Testläufen mit der Hardware teilweise enorme Schwankungen in den gemessenen Größen. Die gewonnenen Erkenntnisse lassen zwar auf grobe Trends schließen mit denen unter Umständen Technologie-Entscheidungen getroffen werden können, aber konkrete Zahlenwerte als Basis für Strategie-Entscheidungen können zum aktuellen Zeitpunkt noch nicht angegeben werden.

Andererseits ist diese Tatsache relativ unkritisch zu bewerten, da im Rahmen des Projektes eine Implementierung mit halbautomatischer Testumgebung geschaffen wurde. Es ist somit möglich nach Abschluss der Entwicklung an der Ethernet-Schnittstelle in kürzester Zeit und mit minimalem Aufwand die gewünschten Ergebnisse zu produzieren.

Alle folgenden Ergebnisse wurden entweder durch Ausführen auf der selben Maschine (im Fall des Emulators) oder durch die Anbindung über 100Mbit-Ethernet im selben lokalen Netzwerk (im Fall der Hardware) gewonnen.

4.2 Latenz und Performance

4.2.1 Übertragung von kleinen Datenpaketen - Latenz

Um die Übertragungsdauer von relativ kleinen Nutzdaten-Objekten bestimmen zu können wurde die Latenz für das Erzeugen und sofortige Schließen von Up- und Download-Sessions herangezogen (Näheres siehe: Kapitel 3.2.1). Es wurden pro Testlauf jeweils 10 Sessions erzeugt und zerstört. Die Ergebnisse der realen Hardware liegen in Form einer Logdatei vor, die Ergebnisse aus dem Emulator sind deutlich weniger breit gestreut und wurden in einem Diagramm ausgewertet.

Reale Hardware

Zu bemerken ist die breite Streuung der Ergebnisse bzw. die »Ausreißer«; zumindest theoretisch sollten sich alle Werte nur um maximal ein paar Millisekunden unterscheiden. Eine Vermutung ist der momentane Stand der Hardware-Implementierung der Ethernet-Schnittstelle; es sieht so aus als käme es hier immer wieder zu verlorenen Paketen und Neu-Übertragungen nach gewissen Timeouts. Allerdings handelt es sich hier um reine Vermutungen - es wurde zunächst keine Zeit zur Lokalisierung der Fehler investiert, da der Hardware-Aufbau momentan nur fliegend vorliegt.

Bei einigen Testläufen scheint es keine groben Ausreißer zu geben - dort beträgt die durchschnittliche Latenz ca. 170msec - da im Test eigentlich die doppelte Latenz gemessen wird, ist mit Werten im Bereich von 80msec bis 90msec zu rechnen.

Auszug aus der resultierenden Log-Datei:

```
 ================================ Starting DataUploadSessionTest ====================
Test started on: 08.07.2010 10:29:06
Going to create and destroy 10 UploadSessions (2*request + 2*response per session).
Filesize (that will be allocated on the service host) is: 12040
Upload Session 1 - Elapsed time was: 0sec 171msec.
Upload Session 2 - Elapsed time was: 0sec 156msec.
Upload Session 3 - Elapsed time was: 0sec 171msec.
Upload Session 4 - Elapsed time was: 4sec 477msec.
Upload Session 5 - Elapsed time was: 13sec 884msec.
Upload Session 6 - Elapsed time was: 0sec 187msec.
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 9sec 250msec.
Upload Session 9 - Elapsed time was: 0sec 187msec.
Upload Session 10 - Elapsed time was: 0sec 187msec.

Average time was: 2sec 884msec.


 ============================== Starting DataDownloadSessionTest ====================
Test started on: 08.07.2010 10:29:35
Going to create and destroy 10 DownloadSessions (2*request + 2*response per session).
Upload Session 1 - Elapsed time was: 4sec 321msec.
Upload Session 2 - Elapsed time was: 0sec 202msec.
Upload Session 3 - Elapsed time was: 5sec 600msec.
Upload Session 4 - Elapsed time was: 4sec 851msec.
Upload Session 5 - Elapsed time was: 4sec 680msec.
Upload Session 6 - Elapsed time was: 4sec 836msec.
Upload Session 7 - Elapsed time was: 4sec 882msec.
Upload Session 8 - Elapsed time was: 4sec 882msec.
Upload Session 9 - Elapsed time was: 4sec 820msec.
Upload Session 10 - Elapsed time was: 3sec 307msec.

Average time was: 4sec 239msec.


 ================================ Starting DataUploadSessionTest ====================
Test started on: 08.07.2010 10:30:35
Going to create and destroy 10 UploadSessions (2*request + 2*response per session).
Filesize (that will be allocated on the service host) is: 12040
Upload Session 1 - Elapsed time was: 0sec 173msec.
Upload Session 2 - Elapsed time was: 0sec 184msec.
Upload Session 3 - Elapsed time was: 0sec 171msec.
Upload Session 4 - Elapsed time was: 0sec 156msec.
Upload Session 5 - Elapsed time was: 0sec 171msec.
Upload Session 6 - Elapsed time was: 0sec 156msec.
```

```
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 0sec 156msec.
Upload Session 9 - Elapsed time was: 0sec 171msec.
Upload Session 10 - Elapsed time was: 0sec 156msec.

Average time was: 0sec 167msec.


 ================================ Starting DataDownloadSessionTest ====================
Test started on: 08.07.2010 10:30:36
Going to create and destroy 10 DownloadSessions (2*request + 2*response per session).
Upload Session 1 - Elapsed time was: 4sec 824msec.
Upload Session 2 - Elapsed time was: 0sec 171msec.
Upload Session 3 - Elapsed time was: 0sec 156msec.
Upload Session 4 - Elapsed time was: 0sec 156msec.
Upload Session 5 - Elapsed time was: 0sec 171msec.
Upload Session 6 - Elapsed time was: 0sec 156msec.
Upload Session 7 - Elapsed time was: 0sec 156msec.
Upload Session 8 - Elapsed time was: 0sec 187msec.
Upload Session 9 - Elapsed time was: 0sec 202msec.
Upload Session 10 - Elapsed time was: 0sec 156msec.

Average time was: 0sec 634msec.


 ================================ Starting DataUploadSessionTest ====================
Test started on: 08.07.2010 10:32:43
Going to create and destroy 10 UploadSessions (2*request + 2*response per session).
Filesize (that will be allocated on the service host) is: 12040
Upload Session 1 - Elapsed time was: 0sec 171msec.
Upload Session 2 - Elapsed time was: 0sec 156msec.
Upload Session 3 - Elapsed time was: 0sec 171msec.
Upload Session 4 - Elapsed time was: 0sec 156msec.
Upload Session 5 - Elapsed time was: 0sec 156msec.
Upload Session 6 - Elapsed time was: 0sec 171msec.
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 4sec 321msec.
Upload Session 9 - Elapsed time was: 7sec 775msec.
Upload Session 10 - Elapsed time was: 4sec 836msec.

Average time was: 1sec 809msec.


 ================================ Starting DataDownloadSessionTest ====================
Test started on: 08.07.2010 10:33:01
Going to create and destroy 10 DownloadSessions (2*request + 2*response per session).
Upload Session 1 - Elapsed time was: 0sec 202msec.
Upload Session 2 - Elapsed time was: 0sec 156msec.
Upload Session 3 - Elapsed time was: 0sec 187msec.
Upload Session 4 - Elapsed time was: 0sec 156msec.
Upload Session 5 - Elapsed time was: 0sec 171msec.
Upload Session 6 - Elapsed time was: 0sec 156msec.
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 0sec 156msec.
Upload Session 9 - Elapsed time was: 0sec 156msec.
Upload Session 10 - Elapsed time was: 0sec 187msec.

Average time was: 0sec 170msec.


 ================================ Starting DataUploadSessionTest ====================
Test started on: 08.07.2010 10:33:37
Going to create and destroy 10 UploadSessions (2*request + 2*response per session).
Filesize (that will be allocated on the service host) is: 12040
Upload Session 1 - Elapsed time was: 0sec 171msec.
Upload Session 2 - Elapsed time was: 0sec 171msec.
```

```
Upload Session 3 - Elapsed time was: 0sec 156msec.
Upload Session 4 - Elapsed time was: 0sec 187msec.
Upload Session 5 - Elapsed time was: 0sec 156msec.
Upload Session 6 - Elapsed time was: 0sec 202msec.
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 0sec 156msec.
Upload Session 9 - Elapsed time was: 0sec 171msec.
Upload Session 10 - Elapsed time was: 0sec 171msec.

Average time was: 0sec 172msec.


 ================================ Starting DataDownloadSessionTest ====================
Test started on: 08.07.2010 10:33:39
Going to create and destroy 10 DownloadSessions (2*request + 2*response per session).
Upload Session 1 - Elapsed time was: 0sec 171msec.
Upload Session 2 - Elapsed time was: 0sec 171msec.
Upload Session 3 - Elapsed time was: 0sec 171msec.
Upload Session 4 - Elapsed time was: 0sec 171msec.
Upload Session 5 - Elapsed time was: 0sec 171msec.
Upload Session 6 - Elapsed time was: 0sec 156msec.
Upload Session 7 - Elapsed time was: 0sec 171msec.
Upload Session 8 - Elapsed time was: 4sec 196msec.
Upload Session 9 - Elapsed time was: 0sec 156msec.
Upload Session 10 - Elapsed time was: 0sec 187msec.

Average time was: 0sec 573msec.
```

Emulator

Jeweils ein *DataUploadSessionTest* und ein *DataDownloadSessionTest* wurden mit dem Service Host am Emulator durchgeführt. Der durchschnittliche Wert für die doppelte Latenz liegt bei ca. 65msec. Dies deckt sich mit der Beobachtung, dass zwischen Emulator und realer Hardware ein Performance-Faktor von ca. 3 liegt. Die Ergebnisse sind in Abbildung 4.1 dargestellt.

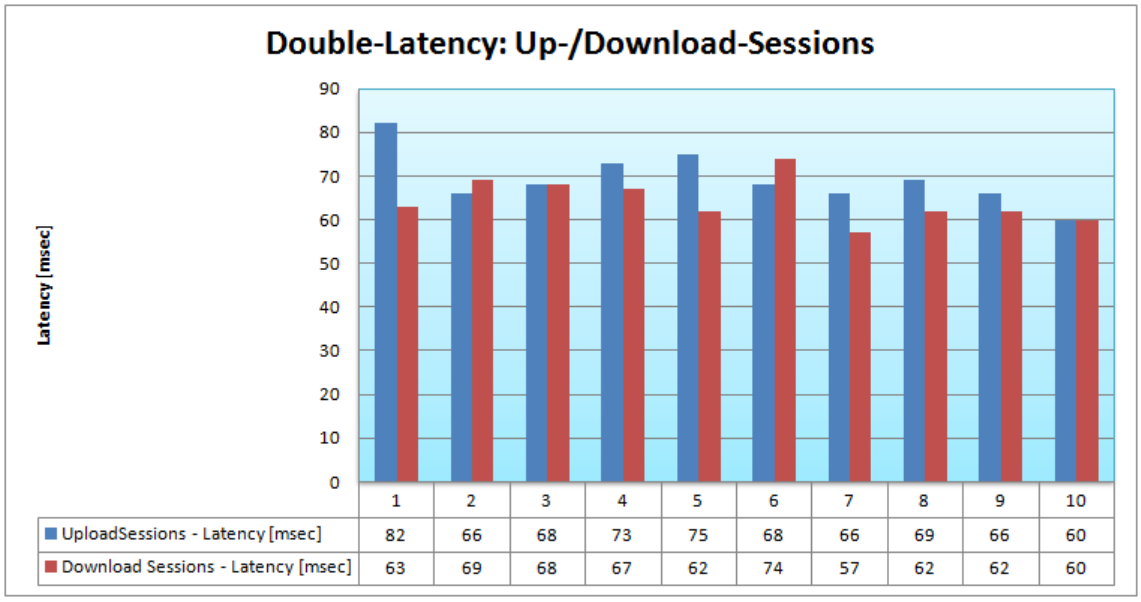

Abb. 4.1: Ergebnisse des Latenztests am Emulator

4.2.2 Übertragung von binären Datenblöcken - Segmentgröße und Datendurchsatz

Um die optimale Segmentgröße für die Übertragung von großen binären Datenblöcken und somit indirekt auch den erreichbaren Datendurchsatz bestimmen zu können, wurden *Upload/Download Segment Size Tests* verwendet (Näheres siehe: Kapitel 3.2.1). In jedem Testlauf wird eine Datei (ca. 240kB) in Segmenten verschiedener Größe vollständig hoch- oder heruntergeladen. Gemessen wird jeweils die Zeit, welche für die Übertragung der gesamten Datei benötigt wurde. Die Ergebnisse der realen Hardware liegen in Form einer Logdatei vor, die Ergebnisse aus dem Emulator sind deutlich weniger breit gestreut und wurden in einem Diagramm ausgewertet.

Reale Hardware

Zu bemerken ist erneut die breite Streuung der Ergebnisse. Beim Ausführen der Tests konnte deutlich verfolgt werden, dass gleich große Segmente unterschiedlich lange zur Übertragung brauchen, obwohl sie theoretisch gleich schnell übertragen werden sollten. Auch hier liegt die Vermutung vor, dass es zu Paketverlusten und Neu-Übertragungen gekommen ist.

Aufgrund der Probleme bei der Durchführung auf der realen Hardware wurden lediglich die drei performantesten Segmentgrößen getestet, wobei hier die Streuung der Ergebnisse der einzelnen Testläufe wesentlich größer ist als der Unterschied zwischen den jeweiligen Segmentgrößen, sodass keine Aussage für eine bestimmte Segmentgröße getroffen werden kann.

Aus den Ergebnissen ist abzusehen, dass die Übertragungsdauer für einen Datei-Upload in günstigen Fällen im Bereich von 12sec bis 16sec liegt. Dieser Wert erscheint insofern realistisch, als dass er ungefähr der dreifachen Übertragungsdauer am Emulator entspricht. Es ergibt sich somit ein zu erwartender Datendurchsatz von ungefähr 18,5 kBps (148kbps).

Für den Download ergeben sich sogar Übertragungszeiten im Bereich von ein bis zwei Sekunden. Auch diese Ergebnisse scheinen im Vergleich mit den Messungen am Emulator realistisch, da am Emulator die Übertragung weniger als eine halbe Sekunde beansprucht. Es ergeben sich somit Übertragungsraten von ca. 120kBps (960kbps).

Auszug aus der resultierenden Log-Datei:

```
================================ Download Segment Size Test ====================
Test started on: 08.07.2010 10:35:45
Downloading 'MicroFramework.png' (241890bytes).

Downloaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 6sec 115msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 39sec 717msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 40sec 373msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'downloading' was: 42000 bytes.
```

```
 ================================ Upload Segment Size Test ========================
Test started on: 08.07.2010 10:37:32
Uploading 'MicroFramework.png' (241890bytes).

Uploaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 39sec 422msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 7sec 82msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 36sec 800msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'uploading' was: 32768 bytes.


 ================================ Download Segment Size Test ====================
Test started on: 08.07.2010 10:40:28
Downloading 'MicroFramework.png' (241890bytes).

Downloaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 16sec 302msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 34sec 742msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 24sec 851msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'downloading' was: 42000 bytes.


 ================================ Upload Segment Size Test ========================
Test started on: 08.07.2010 10:42:01
Uploading 'MicroFramework.png' (241890bytes).

Uploaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 34sec 695msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 22sec 776msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 23sec 494msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'uploading' was: 32768 bytes.
```

```
 ================================ Download Segment Size Test ====================
Test started on: 08.07.2010 10:44:45
Downloading 'MicroFramework.png' (241890bytes).

Downloaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 1sec 762msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 20sec 795msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 12sec 698msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'downloading' was: 42000 bytes.


 ================================ Upload Segment Size Test ========================
Test started on: 08.07.2010 10:45:37
Uploading 'MicroFramework.png' (241890bytes).

Uploaded file with a:

 proposed segment-size of:  42000bytes
 actual segment-size was:  42000bytes, resulting in 6 segments.
 Elapsed time was: 0min 16sec 271msec.

 proposed segment-size of:  32768bytes
 actual segment-size was:  32768bytes, resulting in 8 segments.
 Elapsed time was: 0min 19sec 749msec.

 proposed segment-size of:  16384bytes
 actual segment-size was:  16384bytes, resulting in 15 segments.
 Elapsed time was: 0min 27sec 253msec.

 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 Best segment size for 'uploading' was: 42000 bytes.
```

Emulator

Am Emulator ergeben sich wesentlich gleichmäßigere Ergebnisse, sodass in Abbildung 4.2 jeweils ein Testdurchlauf für Up- und Download dargestellt sind. Außerdem war es möglich, eine größere Anzahl an verschiedenen Segmentgrößen (von 1kB bis ca. 42kB) zu testen.

Aus den Ergebnissen zeigt sich, dass offenbar der Overhead für das Erzeugen von SOAP-/MTOM-Messages wesentlich größer ist, als für das Verarbeiten von großen Messages (mit großen Nutzdaten-Segmenten). Es ist daher sinnvoller wenige große Segmente zu versenden, als die zu übertragende Datei in viele kleine Segmente zu zerlegen.

Außerdem kann aus dem Verlauf der Ergebnisse geschlossen werden, dass im Bereich jenseits von 32kB nur mehr mit mäßigen Performance-Zuwächsen zu rechnen ist. Daher erscheint es aus

aktueller Sicht nicht nötig, die maximale Größe einer Message (64kB) zu erhöhen und somit die Interoperabilität mit Clients anderer Technologien zu gefährden.

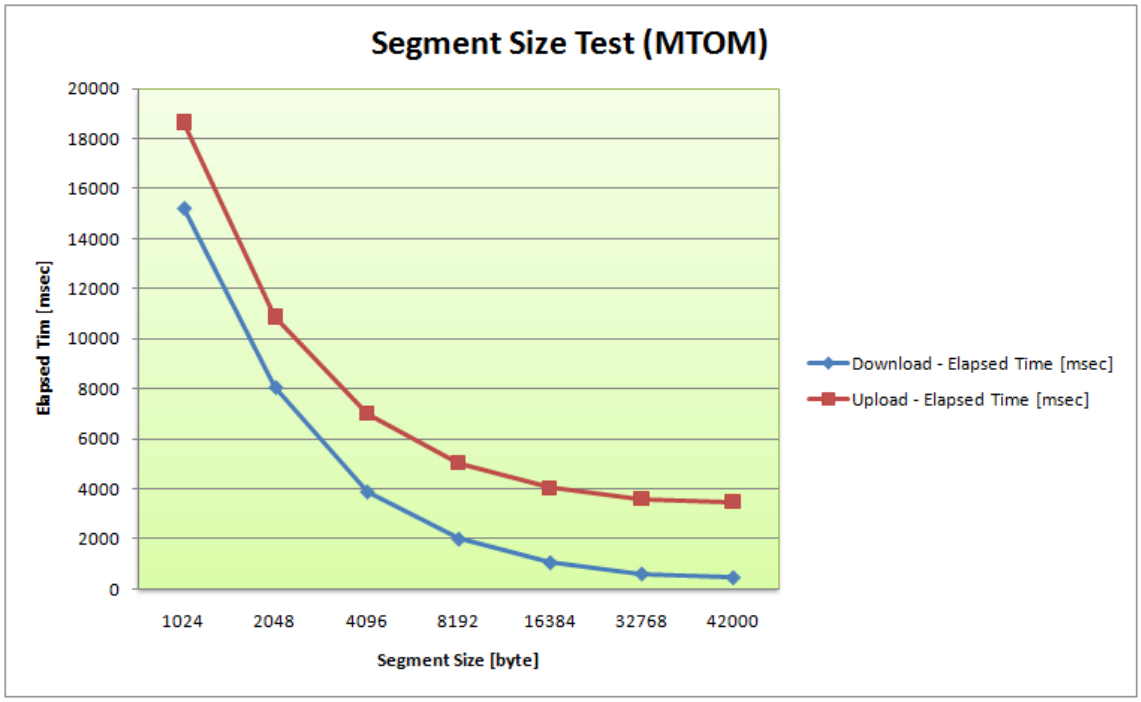

Abb. 4.2: Ergebnisse des Segmentgrößen-Tests (mit 240kB-Datei) am Emulator

4.2.3 MTOM versus SOAP

Da die MTOM-Codierung einen gewissen inhärenten Overhead gegenüber einer reinen SOAP-Übertragung mit sich bringt, war zunächst unklar, wie sich dieser Overhead auf dem Micro Framework auswirkt. WCF hat einen integrierten Mechanismus, welcher bei Nutzdatengrößen von weniger als 1kB trotz MTOM-Codierung, die Daten (mit Hilfe der Base64-Codierung) in die SOAP-Message legt. Es war nicht vorhersehbar, ob für das Micro Framework die selbe Grenze gilt, oder ob sie eventuell niedriger oder höher liegt. Aus diesem Grund wurde die Datenübertragung in zwei Varianten implementiert; mit MTOM oder mit reinem SOAP (Näheres siehe: Kapitel 3.2.1).

Darüber hinaus gab es bis in das letzte Drittel des Projektzeitrahmens Bugs am Micro Framework, welche die Generierung von MTOM-Code nicht zuließen.

Mit der anfänglichen SOAP-Implementierung des *DataAccessService* wurde bereits nach ersten Versuchen im Emulator klar, dass der Upload von Dateien (vor allem in großen Segmenten) äußerst schlechte Ergebnisse liefert. Vermutet wird das Problem in der Umsetzung des XML-Parsers innerhalb des Micro Frameworks. So zeigten sich beim Upload mit der SOAP-Implementierung besonders kleine Segmente als performanteste Lösung. Noch drastischer wurden die Probleme bei Versuchen mit der realen Hardware - der Upload einer 240kB Datei benötigte selbst mit kleinen Segmentgrößen mehr als eine Minute. Ein derartiger Datendurchsatz wäre für die Produkt-reife Implementierung absolut unzulässig.

Durch die Verwendung der MTOM-Codierung konnten sowohl Up- als auch Downloads in adäquaten Zeitspannen realisiert werden.

Reale Hardware

Aufgrund der schlechten Performance bei der Verwendung von SOAP-Messages zur Übertragung von großen Byte-Arrays, konnten mit der Hardware keine brauchbaren Vergleichswerte zwischen den beiden Modi erhoben werden. Um die Problematik zu verdeutlichen wurde eine kleinere Datei (12kB) gewählt, wobei der Upload in einem einzigen Segment erfolgen sollte. Die MTOM-Implementierung benötigte zur Übertragung weniger als eine Sekunde - die SOAP-Variante knapp über eine Minute.

Gerade auf der realen Hardware liegen die beiden Varianten in Bereichen, die sich nicht miteinander vergleichen lassen. Je größer die übertragenen Nutzdaten, desto mehr verschärft sich die Problematik. **Für den Produkt-Einsatz ist die Übertragung von großen Datenblöcken mit SOAP-Nachrichten nicht geeignet!** Es wird ausdrücklich darauf hingewiesen, dass die Problematik nur die Übertragung von »relativ großen« Byte-Arrays betrifft. Für den Austausch von »kleinen« Datenobjekten ergeben sich mit SOAP sehr wohl gute Übertragungszeiten (siehe Ergebnisse der Latenztests; Kapitel 4.1). Die schlechte Performance wird also nicht durch die Verwendung von SOAP oder das XML-Parsen an sich verursacht, sondern lediglich durch das Verarbeiten von großen Byte-Arrays (welche in der SOAP-Message zu einem einzigen langen String werden und daher vermutlich Probleme bei der vorliegenden Parser-Implementierung verursachen).

Emulator

Download (siehe Abbildung 4.3): Es ist zu sehen, dass für besonders kleine Segmente (1kB) die MTOM-Codierung schlechtere Ergebnisse als die Übertragung mit reinen SOAP-Nachrichten liefert. Bereits bei doppelt so großen Segmenten, ist dieser Sachverhalt genau umgekehrt und selbst mit den größten getesteten Segmenten erreicht SOAP nur Werte knapp unter 6 Sekunden; MTOM hingegen Werte knapp unter eine halben Sekunde.

Außerdem zeigt sich auch hier, das die Unterschiede im Bereich großer Segmente wesentlich geringer sind, als sie es beispielsweise für die kleinsten zwei Werte sind.

Upload (siehe Abbildung 4.4): Hier ist deutlich zu erkennen, dass der Upload mittels SOAP im Bereich großer Segmente erhebliche Performance-Probleme verursacht. Aber selbst für kleine Segmentgrößen ist die Übertragung mittels MTOM vorzuziehen.

4.3 Nicht-quantifizierbare Ergebnisse

Neben den messbaren Resultaten, sind im Rahmen der Arbeit eine Reihe von Abschätzungen oder Dokumenten zur Verkürzung der Einlernphase entstanden. Mitunter haben diese Resultate mehr Relevanz als konkrete Zahlenwerte. Es folgt eine kurze Auflistung von weiteren Resultaten:

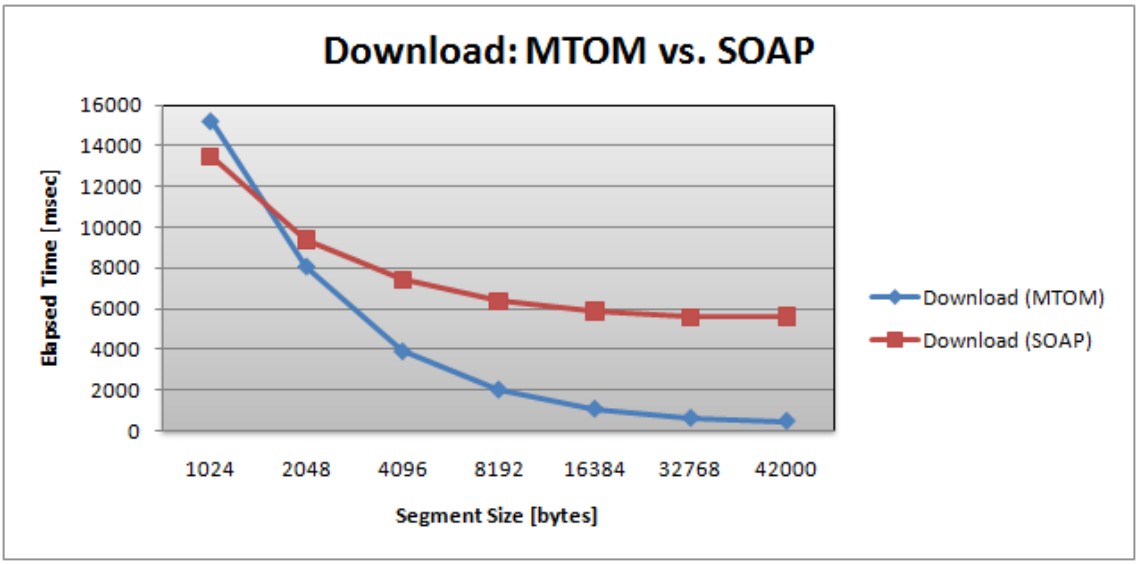

Abb. 4.3: Ergebnisse der Übertragung (Download 240kB) mit SOAP oder MTOM am Emulator

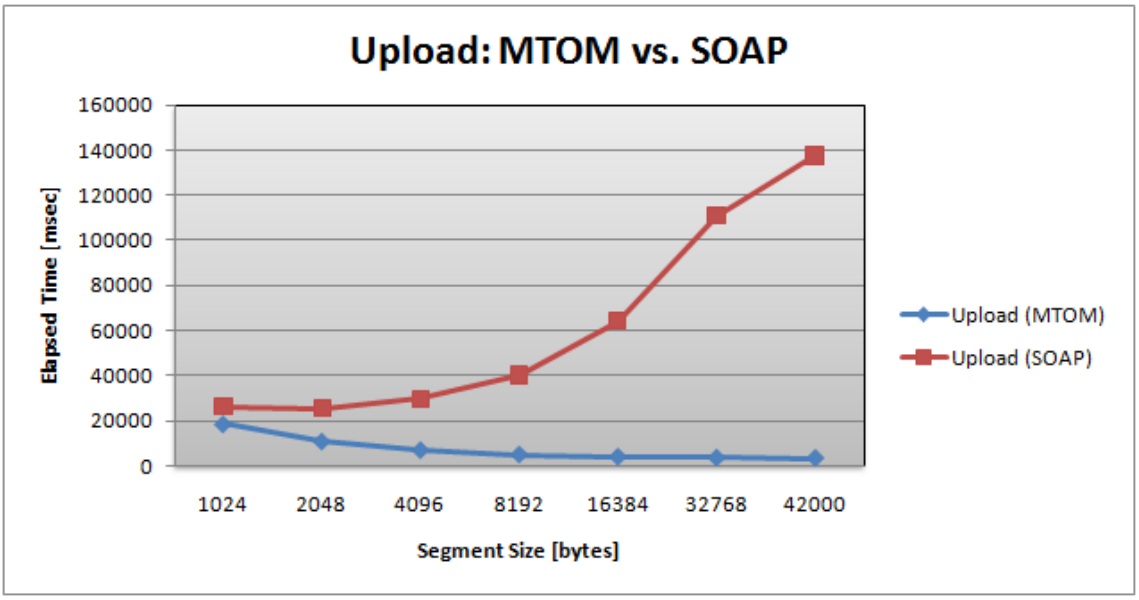

Abb. 4.4: Ergebnisse der Übertragung (Upload 240kB) mit SOAP oder MTOM am Emulator

4.3.1 Security-Recherche

Großes Manko der Webservice-Implementierung am Micro-Framework (DPWS-Stack): die fehlende Implementierung von Security-Mechanismen. Prinzipiell werden bei Webservices zwei Security-Konzepte unterschieden: Transport- und Message-Security. Während bei letzterem Teile der Nachricht verschlüsselt werden, passiert bei der Transport-Security die Verschlüsselung auf niedrigeren Protokoll-Schichten (es wird eine SSL-Verbindung verwendet, bei der im Prinzip die Nachricht als Ganzes verschlüsselt wird).

Es wäre relativ einfach, eine simple Art der Message-Security umzusetzen. Die Problematik

besteht jedoch darin, dass die Umsetzung nach Möglichkeit mit den WS-Security Spezifikationen konform sein sollte. Da diese Vorgaben relativ komplex sind, ist eine derartige Umsetzung mit erheblichem Aufwand verbunden.

Andererseits sind am Micro Framework bereits HTTPS-Sockets (inklusive Ver- und Entschlüsselungs-Funktionen) verfügbar. Um die Transport-Security herzustellen, wäre eine Verwendung dieser Sockets möglich. Allerdings sind auch hier gewisse Standards einzuhalten - vor allem müsste die Implementierung mit einer WCF-Gegenstelle konform sein. Auch diese zweite Version bedarf also eines nicht unerheblichen Implementierungsaufwands.

Da die Übertragung mit SOAP in (lesbarem) Klartext erfolgt und mit Hilfe von simpelsten Tools (nahezu beliebiger Netzwerk-Sniffer) mitverfolgt werden kann, könnten sensible Daten mit einfachen Mitteln ausgelesen oder sogar manipuliert werden. Das Ausarbeiten eines Security-Konzeptes und Umsetzen einer entsprechenden Implementierung hätte in jedem Fall den Rahmen dieser Arbeit überschritten. Aufgrund ihres Ausmaßes und des Aufwandes der zur Behebung nötig ist, wird dir Problematik als kritisch eingestuft. Sollte die Entscheidung zugunsten dem Einsatz von Webservices fallen, wäre ein entsprechendes Security-Konzept mit höchster Priorität auszuarbeiten.

Eine umfassende Einführung in Webservice-Security (inkl. Anwendung mit WCF) ist in [15] zu finden.

4.3.2 Workflow und Best-Practices zur Implementierung von Webservices am Micro Framework

Die Erstellung von Webservices am Micro Framework bzw. der zugehörigen WCF-Gegenstellen bedarf den Einsatz einer Toolchain, die unter anderem zwei Codegeneratoren umfasst. Außerdem sind die jeweiligen Webservice-Implementierungen nicht immer kompatibel, sodass bestimmte Konstrukte vermieden werden müssen. Im Rahmen des Projektes wurden viele dieser »Unstimmigkeiten« oder »Eigenheiten« der jeweiligen Implementierung aufgedeckt und ausführlich dokumentiert (vor allem im Firmen-internen How-To-Wiki). Außerdem wurden Step-By-Step Anleitungen mit zugehörigem Beispielcode verfasst, welche anderen Entwicklern den Einstieg erleichtern und die Einlernphase verkürzen.

Für detaillierte Informationen zur Codegenerierung siehe: Kapitel A.1.

4.3.3 Bug-Submission und zugehörige Bug-Fixes am Micro Framework

Mitunter die wertvollsten Ergebnisse der Arbeit konnten im Bereich der Bug-Behebung am Micro Framework selbst erzielt werden. Die Micro Framework - Community ist sehr überschaubar und oft fehlt es an Ressourcen, wie Codebeispielen oder Tutorials. Glücklicherweise gibt es ein Forum, über das ein direkter Kontakt mit dem Entwickler-Team möglich ist. Genau dort ist auch die Anlaufstelle um eventuelle Bugs melden zu können.

Vor allem die Webservice-Implementierung ist noch relativ neu und erscheint nur oberflächlich getestet - es gibt zwar zu allen Aspekten Minimalbeispiele, die sich reproduzieren und fehlerfrei ausführen lassen; wenn etwas ausführlichere Implementierungen getestet werden, treten allerdings oft Probleme mit dem Framework selbst auf,. Leider sieht es aufgrund der Aktivität in

der Community so aus, als wären höchstens eine Handvoll von Entwicklern damit beschäftigt zwischen **Micro Framework** (DPWS) Geräten und den zugehörigen **WCF**-Gegenstellen zu kommunizieren. Und davon teilt vermutlich weniger als die Hälfte ihr Wissen mit der Community.

Ein Beispiel: Bei der Base64-Codierung gibt es zwei Zeichen in den Konvertierungstabellen welche nicht auf allen Systemen zwingend gleich sein müssen. Die jeweiligen Implementierungen in WCF bzw. am Micro Framework in DPWS unterscheiden sich genau in diesen beiden Zeichen, sodass die erzeugten SOAP-Nachrichten inkompatibel sind.

Es war möglich im Rahmen des Projektes mehrere Bugs oder Interoperabilitäts-Probleme (zwischen DPWS und WCF) zu finden und zu melden. Bereits in der letzten Beta-Release-Serie des Micro Frameworks (4.1 Beta bis 4.1 Beta Refresh #3) wurden einige dieser Bugs behoben. Unter anderem war die Generierung von MTOM-Code aus WSDL-Dateien, die von einem WCF-Service stammen, in der ursprünglichen Version nicht möglich. Aufgrund einer detaillierten Fehlerbeschreibung konnte der Bug von den Entwicklern behoben werden.

Zwar ist der gesamte Sourcecode-Code des Frameworks verfügbar, wodurch es theoretisch möglich wäre, sämtliche Bugs selbst zu beheben - im Sinne einer Verwendung des Micro Frameworks zur kommerziellen Produktentwicklung ist das aber nicht.

Bis zum finalen Release der Version 4.1 war die Webservice-Implementierung auf einem Stand bei dem zwar immer noch im generierten Code nachgearbeitet werden muss - aber prinzipiell alle Anwendungsfälle bedient werden können. In einem Produkt-reifen Einsatz ist die Manipulation von generiertem Code äußerst unerwünscht. Mit der Veröffentlichung des *.NET Micro Framework 4.1* am 19.07.2010 wurden alle im Projekt bekannten Probleme behoben, sodass ein Nacharbeiten im generierten Code nun nicht mehr nötig ist.

Im Sinne der Vorarbeit für die Produkt-reife Umsetzungsphase ist das Aufdecken und Mitwirken an der Behebung derartiger Bugs ein wertvoller Beitrag um *»den Weg für zukünftige Entwicklungen zu ebenen«*.

Das Forum ist erreichbar unter: [7]

4.3.4 Testbed

Nicht zuletzt wurde eine Testumgebung mit einem Projekt mit realistischen Größenordnungen (Abdeckung von Anwendungsfällen, welche über die ausgelieferten Minimalbeispiele hinausgehen) geschaffen. Zwar sind die daraus gewonnenen Ergebnisse noch mäßig aussagekräftig, aber sobald die Entwicklung der Hardware entsprechend vorangeschritten ist, können die gewünschten Resultate quasi »auf Knopfdruck« erzeugt werden.

5 Diskussion

5.1 Bugs und Interoperabilität

Da das .NET Micro Framework bereits zu Beginn des Projektes in seiner vierten Version (4.0) vorlag, war mit einem Reifegrad zu rechnen, welcher den Einsatz zur Produktentwicklung ermöglichen würde. Gerade im Bereich der Webservices konnten diese Erwartungen nur teilweise erfüllt werden. Im Projektverlauf traten zwar keine unlösbaren Probleme auf und die benötigte Funktionalität war in allen Punkten implementiert. Leider kam es aber laufend zu Problemen, welche sich in vielen Punkten auf die fehlerhafte Implementierung innerhalb des Micro Frameworks zurückführen ließen. Es entstand der Eindruck, dass hier nur oberflächlich getestet wurde.

Mit der Version 4.0 wurde auch der Sourcecode-Code des Frameworks öffentlich zugänglich. Diese Änderung der Strategie, brachte vermutlich auch eine Kürzung der Mittel für das Entwicklerteam mit sich. Mehrfach wurde die verstärkte Einbeziehung der Community zur Weiterentwicklung des Frameworks angesprochen. Es traten auch Befürchtungen auf, wonach sich Microsoft aus der Weiterentwicklung des Frameworks zurückziehen würde und das MF quasi »in die Community entlassen« würde. Aufgrund der Aktivität im Entwicklerforum ist aber erkennbar, dass es zumindest noch ein (sehr engagiertes) Kern-Team bei Microsoft gibt, welches die Weiterentwicklung vorantreibt und den Support trägt. Darüber, ob dieses Team mit den nötigen finanziellen bzw. personellen Ressourcen ausgestattet ist, kann aber lediglich spekuliert werden.

Besonders deutlich wird die Problematik im Bereich der Kommunikation zwischen einem **MF**-Device und einer **WCF**-Gegenstelle via Webservices. Offenbar gab es diesen Anwendungsfall bis zur Version 4.0 des Micro Frameworks (zumindest von Seiten der Entwickler aus) nicht. Erst nachdem sich entsprechende Fragen im Forum häuften, reagierte man kurzfristig mit der Veröffentlichung von Minimalbeispielen für die 4.1 Betaversion. Im Zuge der Arbeit traten die meisten Bugs im Bereich dieser Interoperabilität auf, wobei viele Probleme durch relativ offensichtliche Fehler verursacht wurden, welche mit vertretbarem Aufwand lösbar waren. Nach der Meldung dieser Probleme wurden sie meist relativ schnell von offizieller Seite aus behoben und fanden sich im nächsten Zwischenrelease des Frameworks nicht mehr. Ganz klar ist aber, dass es die meisten dieser Probleme gar nie in einen Release hätten schaffen dürfen. Wenn sowohl Client als auch Service-Host auf dem MF ausgeführt wurden, kam es zu wesentlich weniger Problemen; offenbar wurde zumindest hier etwas mehr getestet.

Ein Großteil des Projektes wurde entgegen ersten Erwartungen mit dem Finden, Dokumentieren und vorläufigen Beheben von diversen Bugs und Interoperabilitäts-Problemen verbracht. Im Nachhinein ist dies trotzdem als wertvoll für die zukünftige Entwicklung mit dem Micro Framework anzusehen - wenn Probleme im Forum entsprechend gemeldet werden, ist das Team um deren Behebung äußerst bemüht; ansonsten sieht es aber so aus als wären aus Sicht des Teams die Notwendigkeit oder die entsprechenden Ressourcen für ausführliche Tests nicht gegeben.

Der momentane Stand sieht folgendermaßen aus: Alle Probleme, welche im Rahmen der Projektarbeit gefunden wurden, konnten mittlerweile vom MF-Team oder vorübergehend selbst gelöst werden. Allerdings war es im Projektverlauf mehrfach so, dass behobene Probleme von neuen Problemen abgelöst wurden - meist dadurch, dass erst durch das Beheben bestimmter Probleme komplexere Anwendungsfälle, welche dann neue Probleme aufzeigten, möglich wurden. Immerhin deckt der aktuelle Projektstand eine breite Reihe von Anwendungsfällen ab, sodass die Entwicklung in Produkt-reifer Qualität durchaus plausibel erscheint. Gleichzeitig ist aber aufgrund der gesammelten Erfahrungen damit zu rechnen, dass zumindest neue Interoperabilitätsprobleme auftreten werden (zum Beispiel wurde der WS-Eventing-Mechanismus im Rahmen des Projektes nicht getestet; ein funktionierendes Minimalbeispiel dazu gibt es). Es ist zwar schwierig, derartige Umstände in die Projektplanung miteinzubeziehen, aber ein Mindestmaß an Puffer-Zeiten sollte vorgesehen werden.

Auch wenn die aktuelle Implementierung gewisse Risiken mit sich bringt, stellt sich die Frage nach möglichen Alternativen. Einen gesamten Webservice-Stack selbst zu implementieren würde zu viele Ressourcen in Anspruch nehmen und kommt daher nicht in Frage - der Verzicht auf Webservices würde die Umsetzung einer eigenen Protokollstruktur erfordern, wodurch sich der Aufwand zum Großteil nur verschiebt (anstatt Webservices zu verwenden und eventuelle Bugs zu beheben wird ein eigenes Protokoll entwickelt, das zwar wesentlich simpler sein kann, aber ebenso getestet gehört). Außerdem sind für letzteren Fall die Kosten für eventuelle Weiterentwicklungen und zukünftige Anforderungen schwer abschätzbar - Webservices stellen einen universellen, generischen Mechanismus dar, der nicht an irgendwelche Datentypen oder Kommunikationskanäle gebunden ist. Die Entwicklung einer eigenen Umsetzung, könnte bestenfalls einen Teil dieses Umfangs abdecken.

5.2 Strategien zur Übertragung von großen Datenmengen

Prinzipiell sind Webservices kein Mechanismus, der entwickelt wurde, um große Datenmengen zu übertragen. Vor allem durch die Übertragung im XML-Format ergibt sich ein inhärenter Overhead. Um die Problematik abzuschwächen, wurde die MTOM-Codierung eingeführt, doch auch dort werden nicht nur Nutzdaten übertragen. Es stellen sich Fragen wie: Sind bei der Übertragung kritische Zeitkonstanten einzuhalten? Können die Daten zerlegt werden oder muss der gesamte Datenblock am Stück übertragen werden? Geht es nur um die Übertragung eines Datenblocks, oder soll eine gewisse Semantik in Form von Metadaten mitübertragen werden?

In vielen Fällen ist es daher sinnvoller auf ein spezialisiertes Protokoll wie zum Beispiel FTP zu setzen. Wenn trotzdem Webservices verwendet werden sollen, kann beispielsweise statt dem Datenblock nur ein Verweis auf die Adresse (URL) das Datenblocks angegeben werden - von dort kann er dann per FTP heruntergeladen werden. Im vorliegenden Projekt würde dies jedoch voraussetzen, dass für das Micro Framework ein simpler FTP-Server implementiert wird. Eine andere Möglichkeit wäre es die Daten über ein simples Protokoll auf Socket-Ebene zu übertragen. Allerdings ist es dazu nötig, die entsprechenden Ports freizuschalten, was gerade im Consumer-Markt oftmals ein Problem darstellt. Bei der Verwendung von Webservices kann die gesamte Kommunikation über Port 80 abgewickelt werden - die Firewall-Problematik tritt hier nicht auf.

Wenn keine harten Echtzeit-Anforderungen bestehen, können die Daten prinzipiell auch per Webservice übertragen werden. Aus mehreren Gründen empfiehlt es sich dabei, die Daten in kleinere Segmente zu zerlegen. Einerseits muss bei einer fehlerhaften Übertragung nicht die gesamte Datenmenge neu übertragen werden. Außerdem werden Übertragungsstrukturen weniger stark beansprucht - üblicherweise werden Daten gepuffert übertragen - wenn die Segmente kleiner sind, können auch die zugehörigen Puffer kleiner ausfallen. Es könnte auch sein, dass der Empfänger nur einen Teil der Daten benötigt, um zu entscheiden, ob die gesamte Datenmenge übertragen werden muss oder nicht. Gerade im Hinblick auf die limitierten Ressourcen am Micro Framework macht die Zerlegung in kleinere Datensegmente Sinn.

WCF bietet die Möglichkeit eines sogenannten *Streaming-Bindings* - dabei werden die Daten nicht im herkömmlichen Sinn übertragen sondern gestreamt. Ein entsprechendes Pendant am Micro Framework gibt es leider nicht, sodass diese Funktionalität zunächst zu implementieren wäre.

Zuletzt stellt sich die Frage, ob der Server aktiv oder passiv arbeiten soll. Im aktiven Modus gibt der Server vor, wann ein (weiteres) Segment übertragen wird - dies ist jedoch mit Webservices nicht ganz einfach zu realisieren, da der Server üblicherweise nur auf Requests reagiert. Eine Möglichkeit wäre zum Beispiel die Nutzung von WS-Events.

Im passiven Modus muss der Client die jeweiligen Segmente vom Server einzeln anfordern. Das erlaubt dem Client seine Auslastung relativ genau zu steuern, andererseits könnte der Server mit Anfragen überlastet werden. Im Szenario für das Projekt ist aber davon auszugehen, dass relativ selten gleichzeitige Anfragen an der Server gelangen - vielmehr ist es von Relevanz, dass der Client die Auslastung für ihn möglichst günstig wählen kann (um mit einer Vielzahl von Servern geordnet interagieren zu können). Ein Eventing-Mechanismus wäre hier ungeeignet.

Aus diesen Gründen fiel die Entscheidung im Projekt die Datenmengen in kleinere Blöcke zu segmentieren und diese vom Client aus anzufordern (Server ist passiv). Als große Datenmengen werden im Hinblick auf das Micro Framework Dateigrößen von einigen zig Kilobyte bis maximal in den einstelligen Megabyte-Bereich angesehen.

Teile der angeführten Überlegungen sind in ausführlicher Form unter [6] nachzulesen. Weitere Informationen (inklusive Streaming) sind hier [5] zu finden.

5.3 Bewertung der Einsatzfähigkeit

5.3.1 Reifegrad und Weiterentwicklungen

Der Reifegrad des .NET Micro Framework ist schwer einschätzbar, da viele Features als voll funktionsfähig veröffentlicht werden und dies bis auf die Anwendung mit Minimalbeispielen oft nicht sind. Im Hinblick auf die Webservice-Implementierung kann zumindest davon ausgegangen werden, dass in den letzten Wochen einige Fehler behoben wurden und Standard-Anwendungsfälle nun auch funktional sind. Trotzdem ist der Einsatz des MF mit einem gewissen Risiko verbunden.

Andererseits steht hinter dem Framework ein Entwickler-Team und eine mittlerweile relativ aktive Community, welche den Umfang und die Qualität des Frameworks beständig vorantreiben.

Bei einer Eigenimplementierung der benötigten Funktionalität (Ethernet, Sockets, Webservices, etc.) wäre der Wartungs- (und Testaufwand) komplett auf Seiten des eigenen Entwicklungsteams. Durch den Einsatz des Micro Frameworks ist es möglich sich (hauptsächlich) auf die Entwicklung von Applikationen konzentrieren, während durch die Weiterentwicklung des Frameworks immer neue Features und sogar Zielplattformen zur Verfügung stehen.

Das .NET Micro Framework ist nicht komplett Bug-frei, hat aber einen Reifegrad mit dem ein Produkt-Einsatz unter Einplanung eines gewissen Risikos und entsprechender Puffer-Zeiten möglich ist. Bei allen Problemen, welche im Rahmen des Projektes aufgetreten sind, handelte es sich um Bugs, die sich innerhalb von wenigen Mann-Tagen oder weniger lösen ließen. Es ist somit nicht die Rede von substantiellen Fehlern, sondern von »Kleinigkeiten«. Je nach Anforderung innerhalb eines Projektes kann dieser Umstand aber trotzdem zu einer Entscheidung gegen das MF führen.

Im konkreten Fall versucht die Firma sich jedoch langfristig Kompetenzen im Bereich des Micro Frameworks aufzubauen und führt bereits aktive Entwicklungsarbeit im Bereich der nativen Treiber-Entwicklung oder Erweiterungen des Frameworks durch. Das Finden und Beheben von kleineren Problemen ist daher als relativ unkritisch einzustufen.

5.3.2 Performance

Allgemein ist zu sagen, dass durch die Interpretation des .NET Codes ein nicht unerheblicher Performance-Overhead entsteht. Durch die Verfügbarkeit von günstigen Hardware-Plattformen mit entsprechender Leistung kann dieses Problem aber kompensiert werden. Im Gegenzug erhält man mit dem MF Funktionalität, die wohl mehreren Entwickler-Jahren entspricht. Wenn diese Umstände bei der Planung und Technologie-Entscheidung miteinbezogen werden, hat das Micro Framework durchaus berechtigte Anwendungsfälle.

Im konkreten Fall der Webservices stellt sich erneut die Performance-Frage. Viele »kleine Plattformen« verzichten auf eine Webservice-Implementierung, weil vor allem das XML-Parsing ein spürbarer Ressourcenfresser ist. Andererseits ermöglicht die Technologie die Entwicklung von sehr mächtigen eingebetteten Systemen, welche zukunftssicher in eine modulare Gesamtarchitektur eingebunden werden können. Mit den Messungen, welche im Rahmen des Projektes durchgeführt wurden kann die Performance im Bezug auf Datendurchsatz und Latenz genau eingeschätzt werden. Diese Ergebnisse können dann mit den Anforderungen an die Kommunikationsanbindung abgeglichen werden, um eine Technologie-Entscheidung zu treffen.

Mit den bisher bekannten Anforderungen scheint die Technologie im Hinblick auf ihre Performance für den Einsatz zur Produktentwicklung geeignet.

6 Schluss

6.1 Zusammenfassung

Ziel des Projektes war die Technologieabschätzung und Evaluierung von Webservices auf dem .NET Micro Framework an Hand einer *Proof-Of-Concept*-Implementierung. Um zukünftige Anwendungsfälle abdecken zu können, wurde die beispielhafte Implementierung eines Services zum Up- und Download von Dateien umgesetzt. Zur Umsetzung gehört auch ein Testbett mit dem unter anderem Kennzahlen für Latenz- und Datendurchsatz bestimmt werden können.

Im Lauf der Durchführung wurden immer wieder Bugs und Interoperabilitäts-Probleme (mit WCF-Gegenstellen) gefunden. Nach der Dokumentation dieser Bugs im .NET MF Forum konnten, die meisten dieser Probleme vom MF Entwickler-Team behoben werden (siehe: [7]).

Die Ergebnisse der Durchsatz- und Latenzmessungen sind aufgrund von ausständigen Entwicklungsschritten auf Seiten der Hardware vorerst nur qualitativ aussagekräftig. Sie deuten jedoch in die Richtung, dass die gewählten Ansätze für den Einsatz in einem kommerziellen Projekt geeignet sind.

Darüber hinaus war es Ziel des Projektes, die Einarbeitungszeit für weitere Entwickler zukünftig zu verkürzen und gröbere Probleme aufzudecken und zu dokumentieren. Aus diesem Grund sind mehrere *Step-By-Step*-Anleitungen zur Erstellung von Webservices und *How-To-Wiki*-Artikel verfasst worden.

Der Projektverlauf ist durchaus als positiv zu bewerten - obwohl viel Zeit mit dem Finden und Beheben von Bugs im Micro Framework verbracht wurde, wodurch für die eigentliche Implementierung von Services weniger Zeit blieb.

Der Einsatz des .NET Micro Frameworks lässt sich nicht pauschal befürworten oder ablehnen - er muss im Einzelfall unter den gegebenen Rahmenbedingungen geprüft werden. Ebenso ist die Verwendung von Webservices als primäre Kommunikationsschnittstelle unter den gegebenen Anforderungen des jeweiligen Projektes sehr genau zu prüfen. Geht es rein um die Performance, so sind das Micro Framework oder auch Webservices eher ungeeignete Technologien. Wenn andere Faktoren, wie Time-To-Market, Funktionsumfang, Interoperabilität, Fähigkeit zur Adaption an zukünftige Entwicklungen, etc., in Betracht gezogen werden, können diese Faktoren aber oft das Performance-Argument überwiegen.

6.2 Weitere Schritte

Auf Basis der gewonnenen Erkenntnisse liegt es nun an den Entscheidungsträgern sich für oder gegen den Einsatz von Webservices zu entscheiden. Denkbar wären auch hybride Ansätze, bei

denen nur ein Teil der Funktionalität mit Webservices umgesetzt wird.

Sollte die Entscheidung zugunsten der Services fallen, muss dringend ein Security-Konzept erarbeitet werden. Dazu gehört neben der gesicherten Übertragung auch die Möglichkeit zu Authentifizierung.

Darüber hinaus sind weitere Konzepte im Rahmen der Kommunikationsanbindung auszuarbeiten. In Abhängigkeit der konkreten Anwendungsfälle sind Abläufe zur initialen Verbindungsaufnahme, Aufrechterhalten der Verbindung, dem Austausch von Steuerungsdaten und -Variablen, etc. auszuarbeiten.

Sobald die Entwicklungen an der Hardware abgeschlossen sind, müssen sämtliche Testläufe durchgefahren und ausgewertet werden. Auf Basis dieser Resultate können dann Technologie-Entscheidungen getroffen werden und die nächsten Schritte für eine Produkt-reife Umsetzung gesetzt werden.

A Kapitel im Anhang

A.1 Codegeneration

Im Bereich der Webservice-Entwicklung ist es üblich, den Großteil des Codes für Service- oder Client-Implementierungen automatisiert generieren zu lassen. Meist wird dazu die zugehörige WSDL-Datei herangezogen, es gibt aber auch Varianten bei denen der Code direkt aus einer Assebmly erzeugt werden kann. Im Projekt wurden die Webservice-Implementierungen von WCF und DPWS am Micro Framework verwendet. WCF wird mit einem umfangreichen Codegenerator ausgeliefert, welcher sogar direkt in Visual Studio integriert ist. Das Micro Framework ist mit einem Command-Line Tool zur Erzeugung von Webservice-Code ausgestattet.

A.1.1 Erstellung der Web Service Description - WSDL-Datei

Ausgangspunkt der Codegenerierung ist üblicherweise die Beschreibung des Webservices in Form einer WSDL-Datei. Es drängt sich die legitime Frage auf, wie man zu dieser Beschreibung kommt. Wenn es nur darum geht einen Client zu einem bestehenden Service zu implementieren, ist die Datei meist direkt vom Service-Host zu beziehen. Was aber wenn sowohl Service-Host als auch Client geschrieben werden sollen?

Eine Möglichkeit wäre das Erstellen der WSDL »von Hand«, allerdings sind dafür fundierte Kenntnisse über Webservices, SOAP, XML-Schemata (XSD) und deren Struktur und Typen nötig. Aus diesem Grund sind mehrere graphische XSD- und WSDL-Editoren verfügbar, die den Entwickler von einem Großteil der Komplexität kapseln. Beispielsweise gibt es ein Plugin für die Eclipse-IDE, mit dem grafisch WSDL-Dateien erzeugt werden können.

Im Zusammenhang mit WCF gibt es einen anderen Standard-Workflow. Dort wird der *Service-Contract* direkt in Code geschrieben. Im Detail sieht das so aus, dass für die jeweiligen Services, Interfaces geschrieben werden. Im Contract soll keine Funktionalität, sondern nur Information zur Struktur des Services enthalten sein; ein Interface beschreibt ebenfalls nur Struktur und keine konkrete Funktionalität (d.h. die Implementierung). Die im Interface deklarierten Methoden werden zu Operationen des Services; die in den Methoden verwendeten Datentypen müssen allerdings auch in die Beschreibung des Services mitaufgenommen werden - man spricht von sogenannten *Data-Contracts*. Um nun Methoden als Operationen oder Klassen als Teile des Data-Contracts zu deklarieren werden sogenannte Attribute vergeben (für ein Beispiel siehe Kapitel A.2).

Wird der entsprechende Code in ein *WCF-Service-Library Projekt* gelegt, kann es direkt von Visual Studio aus gestartet werden. Daraufhin wird der Service im integrierten Service-Host

gehostet. Praktischerweise bietet jeder WCF-Service standardmäßig die Möglichkeit, die zugehörige WSDL-Datei zu downloaden. Mit Hilfe dieser Datei ist es dann möglich, den Code für das Micro-Framework zu generieren.

A.1.2 Codegeneration für das Micro Framework

Ausgangspunkt für die Codegeneration ist die WSDL-Datei eines Webservices, die wie im vorhergehenden Kapitel beschrieben, bezogen werden kann. Wird das Tool »*MfSvcUtil.exe*« (aus dem Micro Framework SDK) mit der Datei aufgerufen, wird entsprechender Code für die im Service verwendeten Typen (inklusive zugehöriger Serialisierer- und Deserialisierer-Klassen) generiert. Außerdem wird der Code für einen sogenannten **Hosted Service** und einen **Client-Proxy** erzeugt. Mit Hilfe des Client-Proxys kann der Service konsumiert werden - er bietet die angebotene Funktionalität über einfache Methodenaufrufe.

Mit dem Code für den generierten Hosted Service, kann der Service auf einem Micro Framework Device gehostet werden. Sämtliche (Business-)Funktionalität, die nicht in der WSDL-Beschreibung enthalten ist und somit nicht generiert werden kann, wird im generierten Code praktischerweise in einem Interface gekapselt, sodass zur vollständigen Umsetzung lediglich dieses Interface implementiert werden muss.

Um den generierten Service in einen lauffähigen Zustand zu bringen, ist es noch zusätzlich nötig, ihn in eine MF-Applikation einzubetten und diese entweder am Emulator oder auf der realen Hardware auszuführen.

A.1.3 Codegeneration für einen WCF Client

Um den Code für einen WCF-Client zu generieren gibt es prinzipiell zwei Möglichkeiten. Zugrunde liegt aber beiden Möglichkeiten der WCF-Codegenerator mit dem Namen *«SvcUtil.exe«*. Dieser kann einerseits direkt über Visual Studio aufgerufen werden. Dazu gibt es ein kleines User-Interface bei dem die Adresse eines laufenden Services eingegeben werden kann. Wenn der Service in der selben Projektmappe liegt, kann auch direkt danach gesucht werden. Alternativ kann das Tool mit der vom Server bereitgestellten WSDL-Datei aufgerufen werden.

Der erzeugte Code bietet eine Client-Proxy Klasse zur Kapselung der Webservice-Kommunikation. Für den Anwender kann diese Klasse mit ihren Methodenaufrufen verwendet werden, um mit dem Service-Host zu kommunizieren (im Prinzip wie mit Remote Procedure Calls)

Damit der Client allerdings mit einem Micro-Framework Service Host kommunizieren kann, müssen einige spezielle Konfigurationseinstellungen getroffen werden. Beispielhaft sind diese Einstellungen in Kapitel A.3.2 zu sehen.

A.2 Service-Contracts

A.2.1 DataAccessService

```
namespace DataAccessContract
{

    /// <summary>
    /// Interface (contract) for DataAccessService.
    /// Provides operations for up- and downloading data to and from the
        service host.
    /// Use MTOM-encoding (via binding) if possible!
    /// For managing DataAccessSessions use IDataAccessManagementService
    /// </summary>
    [ServiceContract]
    public interface IDataAccessService
    {
        /// <summary>
        /// Retrieves a specific segment of the specified file (specified
            via the session).
        /// </summary>
        /// <param name="retrievalSessionID">Identifier of the
            session</param>
        /// <param name="segmentIndex">Index (or number) of the segment to
            transmit. Must not exceed NumberOfSegments of current
            session.</param>
        /// <returns>DataSegment-object, containing the requested data
            segment.</returns>
        [OperationContract]
        DataSegment DownloadDataSegment(int dataAccessSessionID, int
            segmentIndex);


        /// <summary>
        /// Uploads a specific segment of the specified file (specified
            via the session) to the service host.
        /// </summary>
        /// <param name="segment">DataSegment-object, containing the data
            to be uploaded</param>
        /// <returns>Zero, if file was received successfully.</returns>
        [OperationContract]
        int UploadDataSegment(DataSegment segment);
    }

}
```

List. A.1: Service-Contract für DataAccessService

A.2.2 DataAccessManagementService

```
namespace DataAccessContract
{
    /// <summary>
    /// Interface (contract) for DataAcccessManagementService.
    /// This service is intended to manage data up- and download to and
        from the service host.
    /// It only provides management-operations for managing
        DataAccessSessions and files.
    /// For up- and downloading data, use IDataAccessService.
    /// </summary>
    [ServiceContract]
    public interface IDataAccessManagementService
    {
        /// <summary>
```

```
/// Creates a new DataUploadSession for uploading data to the
    service host.
/// </summary>
/// <param name="fileName">Name or identifier of the file or array
    to be transmitted.</param>
/// <param name="fileSize">Size of file that is being
    uploaded.</param>
/// <param name="proposedSegmentSize">Size of a single segment as
    proposed by the client. Note that the servicehost might not
    accept that segment size. The actual size will be returned by
    the server.</param>
/// <returns>DataAccessSession-object, containing session
    information.</returns>
[OperationContract]
DataAccessSession StartNewDataUploadSession(String fileName, int
    fileSize, int proposedSegmentSize);

/// <summary>
/// Creates a new DataDownloadSession for downloading data from
    the service host.
/// </summary>
/// <param name="fileName">Name or identifier of the file or array
    to be transmitted.</param>
/// <param name="proposedSegmentSize">Size of a single segment as
    proposed by the client. Note that the servicehost might not
    accept that segment size. The actual size will be returned by
    the server.</param>
/// <returns>DataAccessSession-object, containing session
    information.</returns>
[OperationContract]
DataAccessSession StartNewDataDownloadSession(String fileName, int
    proposedSegmentSize);

/// <summary>
/// Retrieves DataAccessSession-object for a given session.
    Session must have been started already.
/// </summary>
/// <param name="retrievalSessionID">Identifier of the
    session.</param>
/// <returns>DataAccessSession-object containing session
    information.</returns>
[OperationContract]
DataAccessSession GetDataAccessSessionInformation(int
    dataAccessSessionID);


/// <summary>
/// Destroys a DataAccessSession on the servicehost.
/// </summary>
/// <param name="retrievalSessionID">Identifier of the
    session.</param>
[OperationContract]
void CloseDataAccessSession(int dataAccessSessionID, bool
    sesssionAborted);

/// <summary>
/// Deletes a file (permanently) from the file storage of the
    service host.
/// </summary>
/// <param name="fileName">Name of the file to be deleted.</param>
[OperationContract]
void DeleteFilePermanently(String fileName);

/// <summary>
/// Deletes all (!) files from the file storage of the service
    host (permanently!).
```

```
        /// </summary>
        [OperationContract]
        void ClearFileStorage();

        /// <summary>
        /// Retrives information about all files that are present on the
            service host's file storage.
        /// </summary>
        /// <returns>FileInfoList object, containing information about all
            files that are currently stored on the service host.</returns>
        [OperationContract]
        FileInfoList GetFileInformationList();

    }
}
```

List. A.2: Service-Contract für DataAccessManagementService

A.2.3 Data-Contract

```
namespace DataAccessContract
{
    /// <summary>
    /// This object will be provided by the servicehost after creating a
        new DataAccessSession.
    /// It provides information for the consuming client as well as a
        reference ID for the session.
    /// </summary>
    [DataContract]
    public class DataAccessSession
    {
        #region fields

        int sessionID;
        int regularSegmentSize;
        String fileName;
        int fileSize;
        int numberOfSegments;
        int excessSegmentLength;
        bool isUploadSession;

        #endregion

        #region properties
        /// <summary>
        /// Unique session identifier for DataAccessSession.
        /// </summary>
        [DataMember]
        public int SessionID
        {
            get { return sessionID; }
            set { sessionID = value; }
        }

        /// <summary>
        /// Size of a single segment in bytes - note that this is set by
            the server and can differ from the client's desiredSegmentSize.
        /// </summary>
        [DataMember]
        public int RegularSegmentSize
        {
            get { return regularSegmentSize; }
```

```
            set { regularSegmentSize = value; }
        }

        /// <summary>
        /// Name of the file to be transmitted.
        /// </summary>
        [DataMember]
        public String FileName
        {
            get { return fileName; }
            set { fileName = value; }
        }

        /// <summary>
        /// Size of the whole file in bytes. This should equal:
            NumberOfSegments * RegularSegmentSize + ExcessSegmentLength
        /// </summary>
        [DataMember]
        public int FileSize
        {
            get { return fileSize; }
            set { fileSize = value; }
        }

        /// <summary>
        /// Number of segments that the file will be split into.
        /// </summary>
        [DataMember]
        public int NumberOfSegments
        {
            get { return numberOfSegments; }
            set { numberOfSegments = value; }
        }


        /// <summary>
        /// Size of the last segment in bytes.
        /// </summary>
        [DataMember]
        public int ExcessSegmentLength
        {
            get { return excessSegmentLength; }
            set { excessSegmentLength = value; }
        }

        /// <summary>
        /// If this is true, the corresponding session is (only!) for
            uploading data to the service host.
        /// If this is false, the sesssion is (only!) for downloading data
            from the service host.
        /// </summary>
        [DataMember]
        public bool IsUploadSession
        {
            get { return isUploadSession; }
            set { isUploadSession = value; }
        }


        #endregion
    }


    /// <summary>
    /// This object represents a specific segment of the file (data) being
        transmitted.
```

```
/// It contains segment-information and the data-payload.
/// </summary>
[DataContract]
public class DataSegment
{
    #region fields

    int sessionID;
    int segmentSize;
    int segmentIndex;
    byte[] data;

    #endregion

    #region properties

    /// <summary>
    /// Identifier of corresponding session.
    /// </summary>
    [DataMember]
    public int SessionID
    {
        get { return sessionID; }
        set { sessionID = value; }
    }


    /// <summary>
    /// Size of current segment.
    /// </summary>
    [DataMember]
    public int SegmentSize
    {
        get { return segmentSize; }
        set { segmentSize = value; }
    }


    /// <summary>
    /// Index of current segment (i.e. number of current segment).
    /// </summary>
    [DataMember]
    public int SegmentIndex
    {
        get { return segmentIndex; }
        set { segmentIndex = value; }
    }


    /// <summary>
    /// Actual data-payload.
    /// </summary>
    [DataMember]
    public byte[] Data
    {
        get { return data; }
        set { data = value; }
    }

    #endregion
}

/// <summary>
/// Class, containing information about files that are currently
    present on the server (service host).
```

```
/// Contains number of files and size of all files as well as an array
    of FileInfo-objects.
/// </summary>
[DataContract]
public class FileInfoList
{
    #region fields

    int numberOfFiles;
    int sizeOfAllFiles;
    FileInfo[] fileInformation;

    #endregion

    #region properties

    /// <summary>
    /// Number of files that are present on the server.
    /// </summary>
    [DataMember]
    public int NumberOfFiles
    {
        get { return numberOfFiles; }
        set { numberOfFiles = value; }
    }

    /// <summary>
    /// Total size of all files that are present on the server.
    /// </summary>
    [DataMember]
    public int SizeOfAllFiles
    {
        get { return sizeOfAllFiles; }
        set { sizeOfAllFiles = value; }
    }

    /// <summary>
    /// Array of FileInfo-objects, each containing the file's name and
        size.
    /// </summary>
    [DataMember]
    public FileInfo[] FileInformation
    {
        get { return fileInformation; }
        set { fileInformation = value; }
    }

    #endregion
}


/// <summary>
/// File-information object. Containing file's name and size.
/// </summary>
[DataContract]
public class FileInfo
{
    #region fields

    int fileSize;
    String fileName;

    #endregion

    #region properties
```

```
        /// <summary>
        /// Size of file.
        /// </summary>
        [DataMember]
        public int FileSize
        {
            get { return fileSize; }
            set { fileSize = value; }
        }

        /// <summary>
        /// Name of file - this is used to identify the file on the server
            (e.g. for a download).
        /// </summary>
        [DataMember]
        public String FileName
        {
            get { return fileName; }
            set { fileName = value; }
        }

        #endregion
    }

}
```

List. A.3: Data-Contract für DataAccessService und DataAccessManagementService

A.3 XML-Konfigurationsdateien

A.3.1 WCF Service Host

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="DataAccessMtomBinding" messageEncoding="Mtom">
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
              maxArrayLength="42000" maxBytesPerRead="4096"
              maxNameTableCharCount="16384" />
        </binding>
      </wsHttpBinding>
    </bindings>
    <services>
      <service name="DataAccessContract.DataAccessService">
        <endpoint address="" binding="wsHttpBinding"
            bindingConfiguration="DataAccessMtomBinding"
            contract="DataAccessContract.IDataAccessService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
            contract="IMetadataExchange" />
        <host>
          <baseAddresses>
```

```
            <add
                baseAddress="http://localhost:8732/Design_Time_Addresses/DataAccessContract/Da
                />
          </baseAddresses>
        </host>
      </service>
      <service name="DataAccessContract.DataAccessManagementService">
        <endpoint address="" binding="wsHttpBinding"
            contract="DataAccessContract.IDataAccessManagementService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
            contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add
                baseAddress="http://localhost:8732/Design_Time_Addresses/DataAccessContract/Da
                />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information,
          set the value below to false and remove the metadata endpoint
              above before deployment -->
          <serviceMetadata httpGetEnabled="True"/>
          <!-- To receive exception details in faults for debugging
              purposes,
          set the value below to true.  Set to false before deployment
          to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>

</configuration>
```

List. A.4: Konfigurationsdatei des WCF Service Hosts der zur Codegeneration benutzt wird

A.3.2 WCF Client (für Kommunikation mit MF Service Host)

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="Soap11AddressingBinding" >
          <textMessageEncoding
              messageVersion="Soap12WSAddressingAugust2004"/>
          <httpTransport/>
        </binding>
      </customBinding>
      <wsHttpBinding>

        <binding name="DataAccessMtomBinding"  openTimeout="00:05:00"
            sendTimeout="00:05:00" messageEncoding="Mtom">
          <security mode="None">
```

```
            <transport clientCredentialType="None" />
            <message clientCredentialType="None"
                negotiateServiceCredential="false"
              establishSecurityContext="false" />
          </security>
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
              maxArrayLength="42000" maxBytesPerRead="4096"
              maxNameTableCharCount="16384" />
        </binding>

        <binding name="DataAccessSoapBinding" openTimeout="00:05:00"
            sendTimeout="00:05:00" messageEncoding="Text">
          <security mode="None">
            <transport clientCredentialType="None" />
            <message clientCredentialType="None"
                negotiateServiceCredential="false"
              establishSecurityContext="false" />
          </security>
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
              maxArrayLength="42000" maxBytesPerRead="4096"
              maxNameTableCharCount="16384" />
        </binding>

        <binding name="DataAccessManagementBinding"
            messageEncoding="Text" openTimeout="00:05:00"
            sendTimeout="00:05:00">
          <security mode="None">
            <transport clientCredentialType="None" />
            <message clientCredentialType="None"
                negotiateServiceCredential="false"
              establishSecurityContext="false" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint binding="wsHttpBinding"
          bindingConfiguration="DataAccessSoapBinding"
        contract="IDataAccessService"
            name="CustomBinding_IDataAccessSoapService" />
      <endpoint binding="wsHttpBinding"
          bindingConfiguration="DataAccessMtomBinding"
  contract="IDataAccessService"
      name="CustomBinding_IDataAccessMtomService" />
      <endpoint binding="wsHttpBinding"
          bindingConfiguration="DataAccessManagementBinding"
        contract="IDataAccessManagementService"
            name="CustomBinding_IDataAccessManagementService" />
    </client>
  </system.serviceModel>
</configuration>
```

List. A.5: Konfigurationsdatei für einen WCF-Client der mit den am Micro Framework gehosteten Services kommuniziert

Literaturverzeichnis

[1] *The Base16, Base32, and Base64 Data Encodings.* http://tools.ietf.org/html/rfc4648, Abruf: 25.07.2010

[2] *Case Study: Home Controls Manufacturer Uses .NET Micro Framework to Create Product Quickly.* http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?CaseStudyID=201472, Abruf: 25.07.2010

[3] *Devices Profile for Web Services (DPWS).* http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01, Abruf: 25.07.2010

[4] *Documentation for the AUG AMI DevKit for the .NET Micro Framework.* http://www.aug-electronics.com/ami/Documentation/tabid/155/Default.aspx, Abruf: 16.07.2010

[5] *Large Data And Streaming.* http://msdn.microsoft.com/en-us/library/ms733742.aspx, Abruf: 19.07.2010

[6] *Large Data Strategies.* http://msdn.microsoft.com/en-us/library/aa480521.aspx, Abruf: 19.07.2010

[7] *.Net Micro Framework Forums.* http://www.netmf.com/Discussion/Forums.aspx, Abruf: 19.07.2010

[8] *.NET Micro Framework Get Started.* http://www.microsoft.com/netmf/about/gettingstarted.mspx, Abruf: 14.07.2010

[9] *OPC Specifications.* http://www.opcfoundation.org/Downloads.aspx?CM=1&CN=KEY&CI=283, Abruf: 25.07.2010

[10] *SOAP Message Transmission Optimization Mechanism.* http://www.w3.org/TR/soap12-mtom/, Abruf: 25.07.2010

[11] *Windows Communication Foundation.* http://msdn.microsoft.com/en-us/netframework/aa663324.aspx, Abruf: 14.07.2010

[12] Hilbrich, Robert: *Das Leistungspotential von DPWS für Service-orientiertes Ubiquitäres Computing*, HUMBOLDT-UNIVERSITÄT ZU BERLIN, Diplomarbeit, 2009

[13] Kühner, Jens: *Expert .NET Micro Framework, Second Edition.* Apress, 2009

[14] Mahnke, W. ; Leitner, S.H. ; Damm, M.: *OPC Unified Architecture.* Springer, 2009

[15] Meier, J. D. ; Farre, C. ; Taylor, J. ; Bansode, P. ; Gregersen, S. ; Sundararajan, M. ; Boucher, R.: *Improving Web Services Security.* Microsoft Corporation, 2008

[16] Thompson, D. ; Miles, R. S.: *Embedded Programming with the Microsoft .NET Micro Framework*. Microsoft Press, 2007