UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software - Technologie

Proposal für die Diplomarbeit

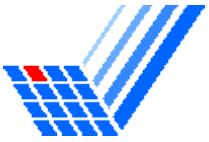
# Konzeption und Entwicklung eines Action/Event-Mechanismus zur Kommunikation mit mobilen Endgeräten

Stefan Göbel

Gutachter:

Prof. Dr. Volker Gruhn
Lehrstuhl für Angewandte Telematik/e-Business
Universität Leipzig

Prof. Dr. Heiko Krumm
Lehrstuhl für praktische Informatik
Universität Dortmund



Draft 0.4 24.04.2020

# Inhaltsverzeichnis

# 1 Einleitung

Die Diplomarbeit mit dem Titel „Konzeption und Entwicklung eines Action/Event-Mechanismus zur Kommunikation mit mobilen Endgeräten“ entsteht im Rahmen des Forschungsverbundprojektes „Mobile Spedition im Web“ (SpiW). Dieses Proposal soll dazu dienen, den Umfang der Arbeit festzulegen, und schon zu einer vorläufigen Gliederung der Diplomarbeit zu gelangen.

Zuerst werden Zweck und Ziele des SpiW-Projektes kurz vorgestellt. Danach wird die Architektur des SpiW-Kommunikationssystems erläutert, bevor die Aufgabenstellung der Diplomarbeit folgt. Um zu einem Lösungsansatz zu gelangen, wird die Aufgabe in mehrere Teilprobleme zerlegt, die kurz beschrieben werden. Außerdem werden die einzusetzenden Mittel genannt. Schließlich wird die Vorgehensweise der Diplomarbeit geplant, einschließlich eines groben Zeitplans.

## 1.1 SpiW

Das Forschungsverbundprojekt „Mobile Spedition im Web“ (SpiW), das vom Bundesministerium für Bildung und Forschung (BMBF) gefördert wird, hat das Ziel, die Auswirkungen des elektronischen Geschäftsverkehrs auf Arbeits- und Unternehmensorganisation von mittelständischen Speditionsunternehmen zu erforschen. Erreicht werden soll eine Verbesserung der Arbeitsprozesse zwischen Spediteuren, Disponenten, Fahrern und Kunden zur Steigerung der Wettbewerbsfähigkeit. Der Fokus des zu entwickelnden SpiW-Kommunikationssystems liegt auf der Verbesserung der Kommunikation zwischen Disponenten und Fahrern. Die Fahrer werden mit mobilen Endgeräten ausgerüstet, mit denen sie Informationen mit dem Kommunikationssystem austauschen können. Das System verfügt außerdem über Schnittstellen zu den Speditionssystemen der Disponenten.

## 1.2 Architektur des SpiW-Kommunikationssystems

Im Wesentlichen besteht das SpiW-Kommunikationssystem aus den Teilen Server, Klienten und Legacy-Systemen.

Der Server seinerseits enthält die fachliche Geschäftslogik, d.h. zum einen die Geschäftsobjekte, die er in Form von Services bereitstellt (z.B. Web Services), zum anderen einen Workflow-Server, der die Arbeitsprozesse steuert. Eine Datenbank ist für die Persistenz zuständig. Der letzte Teil ist der Kommunikations-Server. Er regelt den Datenaustausch mit den Klienten und den Legacy-Systemen.
Es gibt zwei Arten von Klienten: die mobilen Klienten, die auf mobilen Endgeräten laufen, und die Dispositions-Klienten, die auf stationären Geräten betrieben werden. Das Datenübertragungsformat zu den Klienten ist XML, das Übertragungsmedium ist entweder LAN (Dispositions-Client) oder Mobilfunk (Mobiler Client). Es werden nicht nur Geschäftsobjekte vom Server zu den Klienten und zurück übertragen, sondern auch Ereignisse (Events), die jeweils zu einer Aktion (Action) führen. Dieses Prinzip wird als Event/Action-Mechanismus bezeichnet.

Da auch Legacy-Systeme (z.B. ein Speditionssystem) angebunden werden, müssen Schnittstellen zu diesen Systemen existieren. Sie bedienen sich auch dem Event/Action-Mechanismus, d.h. an den Schnittstellen treten Ereignisse auf, die dann zu Aktionen in den Legacy-Systemen führen. Dadurch ist eine große Flexibilität gewährleistet: Änderungen beim Legacy-System

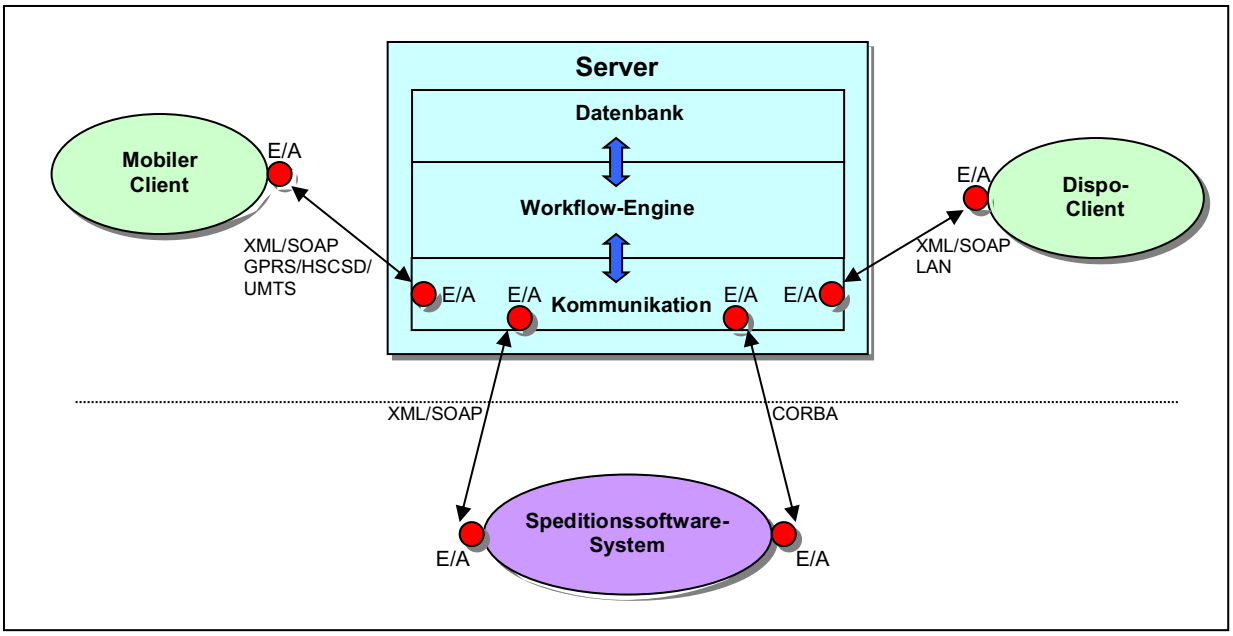



**Abbildung 1:** Architektur des Kommunikationssystems

führen nicht zu Änderungen an der Schnittstelle; mehrere Legacy-Systeme können sich eine Schnittstelle teilen. Schnittstellen können mit verschiedenen Techniken realisiert werden, z.B. über CORBA-Services oder über das Simple Object Access Protocol (SOAP).

# 2 Aufgabenstellung

Die Kommunikation zwischen dem Server und den Klienten erfolgt durch den Austausch von XML-Daten. Es sollen komplexe Objekte übertragen werden, zum einen Geschäftsobjekte, zum anderen Ereignisse (Events), die dann auf dem Server bzw. auf den Klienten Aktionen auslösen. Die Geschäftsobjekte und Events müssen vor der Übertragung in ein XML-Format umgewandelt und nach der Übertragung wieder zurück gewandelt werden. Dieser Mechanismus, der als „Data Binding“ bezeichnet wird [Mc02], muss sowohl auf dem Server als auch auf den Klienten zur Verfügung stehen.
In der Arbeit soll ein Konzept erstellt werden, durch das Ereignisse und Geschäftsobjekte mit XML beschrieben werden können. Das Konzept muss flexibel erweiterbar sein, um auf geänderte Rahmenbedingungen (neue Ereignisse und Geschäftsobjekte) eingehen zu können. Dann muss überlegt werden, wie ein solches XML-Dokument erzeugt werden kann bzw. wie aus einem XML-Dokument Objekte erzeugt werden können (Data Binding). Die tatsächliche Übertragung von XML zum Server bzw. zu den Klienten ist auch Teil dieser Arbeit. Außerdem soll bei auftretenden Events eine Ableitung erfolgen, welche Aktionen ausgeführt werden müssen (Mapping). Die prototypische Realisierung, durch die die Praktikabilität des Gesamtkonzptes nachgewiesen werden soll, muss sich problemlos in die SpiW-Architektur einfügen können.

# 3 Lösungsansatz

## 3.1 Zerlegung in Teilprobleme

Die Aufgabenstellung lässt sich in vier Teilprobleme zerlegen:

a) Event/Action-Mechanismus
b) Beschreibung des XML-Datenformats
c) Data Binding
d) Übertragung der XML-Dokumente

### 3.1.1 Event/Action-Mechanismus

Die Idee ist, dass alle zu übertragenden Daten in Events gekapselt werden, also sowohl auftretende Ereignisse wie Störfälle oder erledigte Transporte als auch Geschäftsobjekte. Dadurch stellt sich die Frage nach der angemessenen Granularität der Events:

a) Zu jedem Parameter jeder Klasse eine Event-Klasse (feine Granularität)

Vorteile:
- Einfache Klassen
- Leicht nachzuvollziehen

Nachteile:
- Große Anzahl von Klassen
- Viele Events = Viele Listener
- Neuer Parameter = Neues Event = Neuer Listener (schwierig zu erweitern)

b) Zu jeder Klasse eine Event-Klasse (mittlere Granularität)

Vorteile:
- Neue Parameter bedingen keine neuen Event-Klassen
- Leicht nachzuvollziehen

Nachteile:
- Relativ viele Klassen
- Neue Klasse bedingt neuen Listener (schwierig zu erweitern)

c) Eine globale Event-Klasse (grobe Granularität)

Vorteile:
- Man braucht nur einen Listener
- Neue Events bedingen keine neuen Listener
- Gut erweiterbar durch Mapping-Tabelle

Nachteile:
- Nicht schön im objektorientierten Sinne
- Komplizierte Event-Klasse
- Übersichtlichkeit könnte leiden

Man muss beachten, dass Geschäftsobjekte weitere Geschäftsobjekte als Attribute in sich tragen können. Deswegen könnte die optimale Lösung folgendermaßen aussehen: es gibt nur eine Event-

Klasse, die die Attribute der Ereignisse und Geschäftsobjekte in sich kapselt. Hat ein Geschäftsobjekt noch Unterobjekte, wie z.B. ein Transport mehrere Transportpositionen hat, werden diese Unterobjekte selbst in ein solches Event gekapselt und diese Unterevents werden dann durch das Oberevent aggregiert. Dadurch entsteht eine Baumstruktur, die die Umwandlung von bzw. nach XML erleichtern könnte. Anschließend muss überlegt werden, wie das Event Aktionen auslöst und welche Aktionen es auslöst (Mapping).

### 3.1.2 Beschreibung des XML-Datenformats

Es gibt zwei Methoden zur Modellierung eines XML-Datenformats: Dokumenttyp-Definitionen (DTDs) [Ra01] und XML-Schema [Vl02]. Der erste Schritt besteht darin, diese beiden Alternativen gegenüber zu stellen, um sich dann für eine Methode zu entscheiden. Als nächstes muss dann, unter den Gesichtspunkten Flexibilität und Erweiterbarkeit, das tatsächliche Format definiert werden.

### 3.1.3 Data Binding

Data Binding ist ein Verfahren, das ein XML-Dokument parst, die darin enthaltenen Daten extrahiert und daraus Objekte generiert. Data Binding vereinigt drei unterschiedliche Prozesse: Klassengenerierung, Unmarshalling und Marshalling. Die Klassengenerierung ist ein Prozess, der eine Menge von XML-Beschränkungen (wie in DTDs oder XML Schemas) nimmt und daraus C#-Klassen erzeugt. Das Vorgehen ist einfach: XML-Beschränkungen sind äquivalent zu C#-Klassen-Definitionen; sie definieren die Art und Weise, in der Daten repräsentiert werden. Auf der anderen Seite ist ein XML-Dokument äquivalent zu einer Instanz dieser Klassen, da es einfach Daten enthält, die den Kontrakt erfüllen, der durch die Beschränkungen des Dokuments definiert wurde. Das Unmarshalling ist der Prozess der Umwandlung eines XML-Dokuments in die Instanz einer C#-Klasse. Entsprechend ist das Marshalling das genaue Gegenteil. Es bezeichnet den Prozess der Umwandlung eines C#-Objekts und darin enthaltener Objekte in eine XML-Repräsentation [Mc02].

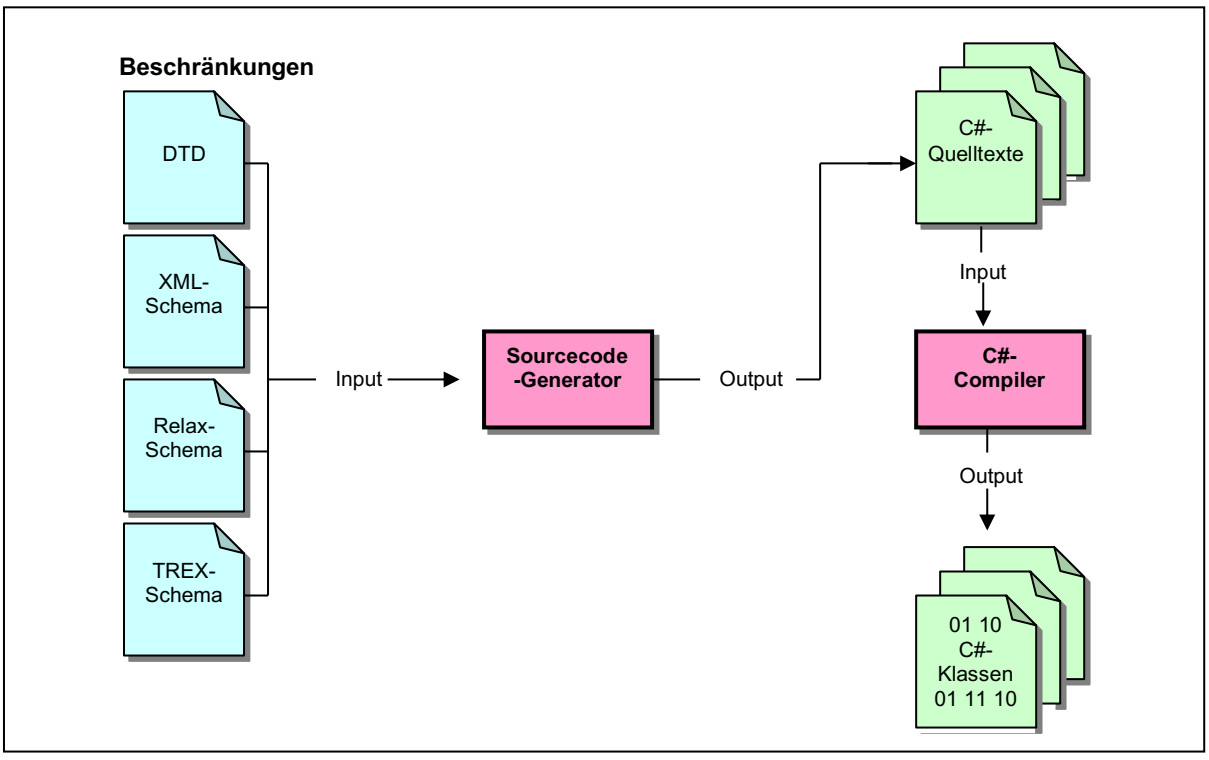


**Abbildung 2:** Klassengenerierung bei XML-Data-Binding [Mc02]

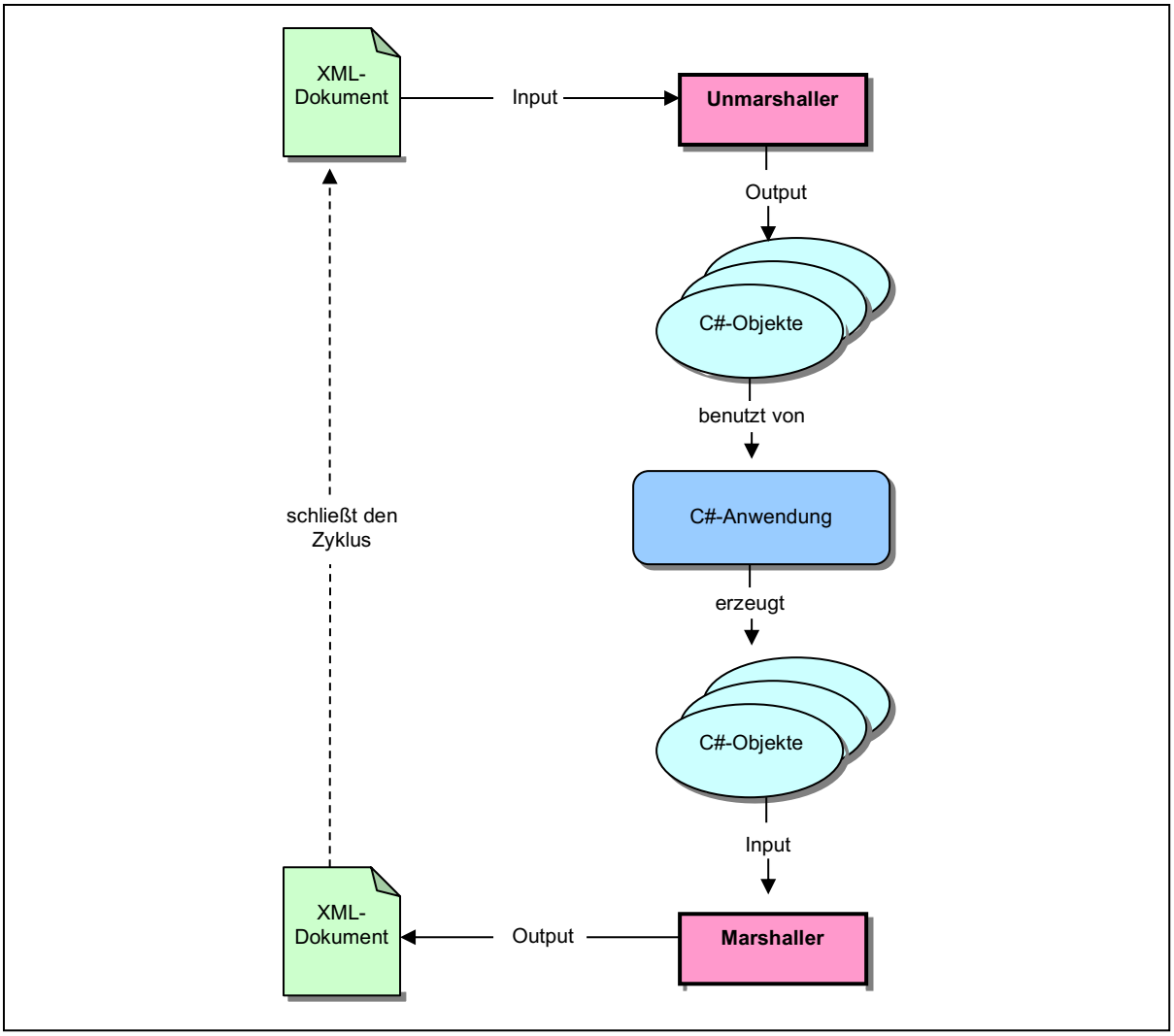



**Abbildung 3:** XML-Data-Binding-Kontrollfluß [Mc02]

In der Arbeit muss ein solches Data-Binding-Verfahren konzeptioniert und prototypisch implementiert werden.

### 3.1.4 Übertragung der XML-Dokumente

Es muss erörtert werden, wie denn die XML-Daten tatsächlich übertragen werden. Bis dato sind zwei Alternativen bekannt: die Kommunikation über .NET Remoting und die Nutzung von Webservices. Der Nachteil von .NET Remoting liegt in der Plattformabhängigkeit. Sowohl Server als auch die Klienten müssen das .NET Framework unterstützen und die benötigten Klassen implementieren. Hier liegt genau der Vorteil von Webservices: sie sind plattformunabhängig und die Methoden eines Webservices können über das Internet aufgerufen werden. Dazu wird eine WSDL-Datei benötigt, die folgende Informationen beinhaltet [Mc02]:

- den Namen des Services
- den Ort, an dem auf den Service zugegriffen werden kann
- die aufrufbaren Methoden
- die Argumenttypen und die Typen der Rückgabewerte für jede Methode

Der Transportmechanismus für die Daten ist das Simple Object Access Protocol (SOAP). Die Problematik bei der Benutzung von Webservices liegt in der Kommunikation von Server zu Klient: Während der Klient beim Server die entsprechenden Methoden aufrufen kann, ist der Server passiv und kann auf den Klienten keine Methoden aufrufen. Deshalb muss ein entsprechender Mechanismus erdacht werden. Denkbar ist z.B. folgendes: Der Klient ruft beim Start asynchron eine Methode des Webservices auf, so dass der Server dann seinerseits die zu übermittelnden Daten als Rückgabewert dieser Methode verschicken kann. Hat der Server aber momentan keinen Bedarf an Kommunikation mit dem Klienten, wird ein Thread erzeugt, der solange läuft, bis wieder etwas verschickt werden soll.

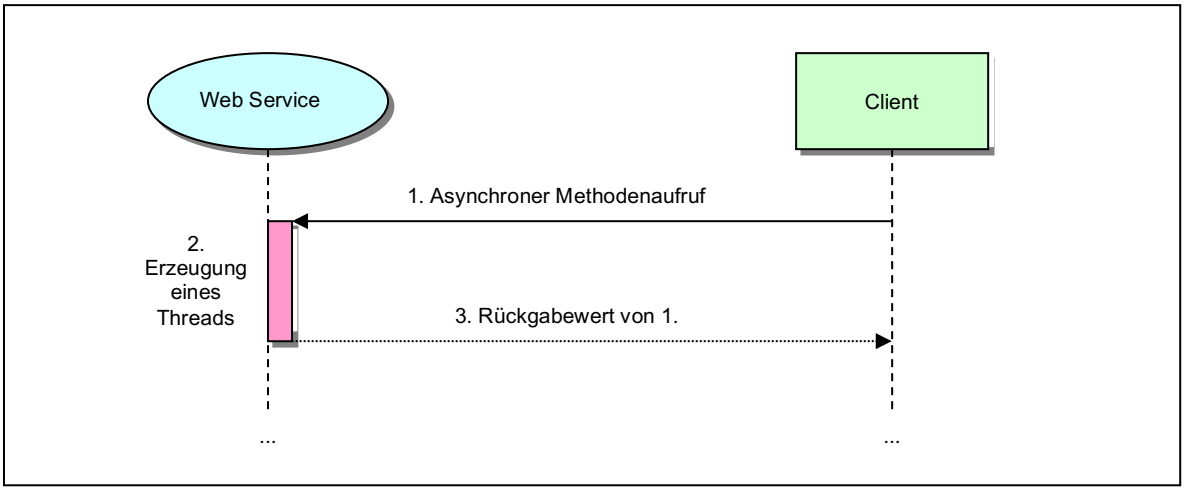



**Abbildung 4:** Mögliche Problemlösung zur bidirektionalen Kommunikation (1)

Eine weitere Möglichkeit ist, dass der Klient alle x Zeiteinheiten die entsprechenden Methoden des Webservices aufruft, um z.B. an die an ihm gerichteten Events zu gelangen.

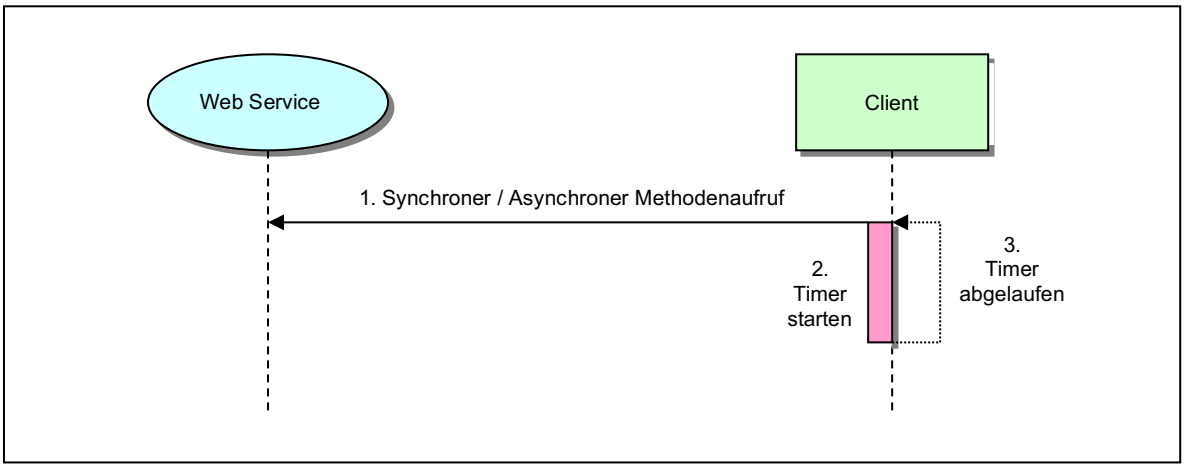



**Abbildung 5:** Mögliche Problemlösung zur bidirektionalen Kommunikation (2)

Es gibt noch weitere Aspekte zur Datenübertragung, die im Rahmen der Arbeit betrachtet werden können. Dazu zählen Verschlüsselung, Datenkomprimierung und Verhalten bei Verbindungsabbruch.

## 3.2 Einzusetzende Mittel

Da diese Arbeit im Rahmen des SpiW-Projektes entsteht, muss sie sich mit den Rahmenbedingungen des Projektes zufrieden geben, insbesondere weil sich das zu entwickelnde System nahtlos in die SpiW-Architektur einfügen soll; jede verfügbare Hard- und Software des SpiW-Projektes kann genutzt werden.
Somit ist die Entwicklungsumgebung sowohl für die Client- als auch für die Server-Komponenten Microsoft Visual Studio .NET. Genutzt wird ebenso das .NET Compact Framework für die mobilen Klienten. Die einzusetzende Programmiersprache ist C#.

# 4 Geplantes Vorgehen

## 4.1 Aufteilung in Arbeitspakete

Aus der Zerlegung in Teilprobleme (siehe 3.1) kann die Aufgabe in folgende Arbeitspakete aufgeteilt werden:

- AP 1: Event/Action-Mechanismus
  - AP 1.1: Granularität diskutieren
  - AP 1.2: Spezifikation der Events / Eventhandler
  - AP 1.3: Mapping von Ereignissen nach Aktionen
  - AP 1.4: Realisierung
  - AP 1.5: Dokumentation
  - AP 1.6: Optimierungen

- AP 2: Beschreibung des XML-Datenformats
  - AP 2.1: DTD vs. XML-Schema
  - AP 2.2: Definition des Datenformats
  - AP 2.3: Optimierungen

- AP 3: Data Binding
  - AP 3.1: Data Binding Frameworks vs. Eigenbau
  - AP 3.2: SAX vs. DOM
  - AP 3.3: Konzeption des Data-Binding-Mechanismus
  - AP 3.4: Realisierung
  - AP 3.5: Dokumentation
  - AP 3.6: Optimierungen

- AP 4: Übertragung der XML-Daten
  - AP 4.1: Verschiedene Möglichkeiten erörtern
  - AP 4.2: Konzeption / Lösung der Probleme bei Web Services
  - AP 4.3: Realisierung
  - AP 4.4: Dokumentation
  - AP 4.5: Optimierungen

- AP5: Sonstiges
  - AP 5.1: Einleitung
  - AP 5.2: Ausblick
  - AP 5.3: Layout
  - AP 5.4: Endkontrolle

Außerdem sollte noch ein Puffer zur Sicherheit in den Zeitplan einfließen.

## 4.2 Zeitplan

Die Definition des XML-Datenformats kann natürlich erst nach der genauen Spezifikation der Events stattfinden, deshalb kommt AP 2 nach AP 1. Danach folgt das Arbeitspaket AP 3, dann AP 4. Kann der Zeitplan eingehalten werden, finden danach die Optimierungen statt, bevor dann in AP 5 die Diplomarbeit beendet wird.
Bei der voraussichtlichen Anmeldung der Diplomarbeit zum 1. April, ergibt sich der 30. September als Abgabetermin (=26 Wochen). Der Zeitplan sieht wie folgt aus:

| Nr. | Vorgangsname | Dauer | Anfang | Ende |
|---|---|---|---|---|
| 1 | AP 1.1 | 5 Tage? | Di 01.04.03 | Mo 07.04.03 |
| 2 | AP 1.2 | 5 Tage? | Di 01.04.03 | Mo 07.04.03 |
| 3 | AP 1.3 | 10 Tage? | Di 08.04.03 | Mo 21.04.03 |
| 4 | Urlaub | 5 Tage? | Di 22.04.03 | Mo 28.04.03 |
| 5 | AP 1.4 | 5 Tage? | Di 29.04.03 | Mo 05.05.03 |
| 6 | AP 1.5 | 5 Tage? | Di 06.05.03 | Mo 12.05.03 |
| 7 | AP 1 fertig | 0 Tage | Mo 12.05.03 | Mo 12.05.03 |
| 8 | AP 2.1 | 5 Tage? | Di 13.05.03 | Mo 19.05.03 |
| 9 | AP 2.2 | 5 Tage? | Di 20.05.03 | Mo 26.05.03 |
| 10 | AP 2 fertig | 0 Tage | Mo 26.05.03 | Mo 26.05.03 |
| 11 | AP 3.1 | 5 Tage? | Di 27.05.03 | Mo 02.06.03 |
| 12 | AP 3.2 | 5 Tage? | Di 27.05.03 | Mo 02.06.03 |
| 13 | AP 3.3 | 5 Tage? | Di 03.06.03 | Mo 09.06.03 |
| 14 | AP 3.4 | 5 Tage? | Di 10.06.03 | Mo 16.06.03 |
| 15 | AP 3.5 | 5 Tage? | Di 17.06.03 | Mo 23.06.03 |
| 16 | AP 3 fertig | 0 Tage | Mo 23.06.03 | Mo 23.06.03 |
| 17 | AP 4.1 | 5 Tage? | Di 24.06.03 | Mo 30.06.03 |
| 18 | AP 4.2 | 5 Tage? | Di 01.07.03 | Mo 07.07.03 |
| 19 | AP 4.3 | 5 Tage? | Di 08.07.03 | Mo 14.07.03 |
| 20 | AP 4.4 | 5 Tage? | Di 15.07.03 | Mo 21.07.03 |
| 21 | AP 4 fertig | 0 Tage | Mo 21.07.03 | Mo 21.07.03 |
| 22 | AP 5.1 | 10 Tage? | Di 22.07.03 | Mo 04.08.03 |
| 23 | AP 5.2 | 10 Tage? | Di 22.07.03 | Mo 04.08.03 |
| 24 | Optimierungen | 15 Tage? | Di 05.08.03 | Mo 25.08.03 |
| 25 | Optimierungen fertig | 0 Tage | Mo 25.08.03 | Mo 25.08.03 |
| 26 | AP 5.3 | 10 Tage? | Di 26.08.03 | Mo 08.09.03 |
| 27 | AP 5.4 | 10 Tage? | Di 26.08.03 | Mo 08.09.03 |
| 28 | Puffer | 15 Tage? | Di 09.09.03 | Mo 29.09.03 |
| 29 | Abgabetermin | 0 Tage | Mo 29.09.03 | Mo 29.09.03 |

# 5 Stand der Arbeit

Bezüglich AP 1.1 und AP 1.2 wurde schon einige Vorarbeit geleistet (siehe 3.1.1). So wurde die optimale Granularität diskutiert und auch schon die Struktur der Eventklasse spezifiziert. Die Struktur der Klasse könnte wie folgt aussehen:

**Net.Spiw.Event.SpiwEvent**

**GetSource()**: die IP-Adresse der Quelle dieses Ereignisses
**GetTarget()**: die IP-Adresse des Ziels, wo dieses Event hingeschickt werden soll
**GetCategoryKey()**: dieser Schlüssel bestimmt, was für eine Art von Ereignis aufgetreten ist (Status, Störung usw.)
**GetSubKey()**: dieser Schlüssel bestimmt, was genau für ein Ereignis innerhalb der oben spezifizierten Kategorie aufgetreten ist (Auftrag geladen, ausgeliefert, Stau usw.)
**GetMessage()**: eine Nachricht für dieses Event
**GetAttributes()**: liefert eine Hashtable mit allen Attributen
**GetObjectType()**: liefert den Typ des gekapselten Geschäftsobjektes

In C# sieht das dann folgendermaßen aus:

```
using System;
using System.Collections;
namespace Net.Spiw.Event.SpiwEvent
{
  class SpiwEvent
  {
    String GetSource();
    String GetTarget();
    String GetCategoryKey();
    String GetSubKey();
    String GetMessage();
    Hashtable GetAttributes();
    String GetObjectType() ;
 }
}
```

Auch bezüglich AP 4 wurde schon Vorarbeit geleistet. So wurden mit .NET Remoting und Web Services erste Erfahrungen gesammelt. Eine wichtige Erkenntnis ist, das .NET Remoting nicht vom .NET Compact Framework unterstützt wird, somit muss die Datenübertragung über Web Services stattfinden.

# 6 Literaturverzeichnis

Folgende Bücher wurden zur Erstellung dieses Dokumentes benutzt:


[Mc02] B. McLaughlin: „Java & XML“, 2. Auflage, O'Reilly, Köln, 2002
[Ra01] E.T. Ray: „Einführung in XML“, 1. Auflage, O'Reilly, Köln, 2001
[Vl02] E. van der Vlist: „XML Schema“, 1. Auflage, O'Reilly, Sebastopol, 2002


Folgende Bücher wurden zur Vorbereitung zur Diplomarbeit gelesen bzw. werden noch gelesen, werden aber für dieses Dokument nicht genutzt:


[Ar01] T. Archer: „ Inside C#“, 1. Auflage, Microsoft Press, Unterschleißheim, 2001
[DeDe01] H.M. Deitel, P.J. Deitel, T.R. Nieto, T. Lin, P. Sadhu: „XML How to Program“, 1. Auflage, Prentice-Hall, New Jersey, 2001
[DeLi96] M. Deininger, H. Lichter, J. Ludewig, K. Schneider: „Studien-Arbeiten“, 3. Auflage, vdf, Zürich; Teubner, Stuttgart, 1996
[DiGa00] K.R. Dittrich, S. Gatziu: „Aktive Datenbanksysteme“, 2. Auflage, dpunkt-Verlag, Heidelberg, 2000
[Ha02] A. Hanisch: „XML mit .NET - Programmierung und Basisklassen“, 1. Auflage, Addison-Wesley, München, 2002
[HaMe01] E.R. Harold, W.S. Means: „XML in a Nutshell“, 1. Auflage, O'Reilly, Köln, 2001
[Mc02a] B. McLaughlin: „Java & XML Data Binding“, 1. Auflage, O'Reilly, Sebastopol, 2002
[MoPo02] C. Morris , A. Polshaw, S. Sivakumar , P. Stanski „.NET Compact Framework“, 1. Auflage, Wrox Press, Birmingham, 2002
[Pl01] D.S. Platt: „Microsoft .NET – Eine Einführung“, 1. Auflage, Microsoft Press, Unterschleißheim, 2001
[ScSt02] K. Scribner, M.C. Stiver: „Applied SOAP: Implementing .NET XML Web Services“, 1. Auflage, Sams Publishing, Indianapolis, 2002
[Ta02] R. Tabor: „Microsoft .NET XML Web Services“, 1. Auflage, Sams Publishing, Indianapolis, 2002
[We02] C. Weyer: „XML Web Service-Anwendungen mit Microsoft .NET“, 1. Auflage, Addison-Wesley, München, 2002
[Wi02] M. Williams: „Microsoft Visual C# .NET Entwicklerhandbuch“, 1. Auflage, Microsoft Press, Unterschleißheim, 2002
[WiWh03] A. Wigley, S. Wheelwright: „Microsoft . NET Compact Framework“, 1. Auflage, Microsoft Press, Redmond, 2003