UNIVERSITÄT KOBLENZ · LANDAU

ist Institut für Softwaretechnik

PegelSuite: Eine Web Service orientierte Umgebung zur Verarbeitung von Pegeldaten

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

Juli 2011

vorgelegt von

Christian Bruckhoff <brchrist@uni-koblenz.de>
Matrikelnummer: 203210045

Betreuer

Prof. Dr. Jürgen Ebert <ebert@uni-koblenz.de>
Dr. Volker Riediger <riediger@uni-koblenz.de>

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Danksagung

Ich möchte mich besonders bei *Prof. Dr. Jürgen Ebert* und *Dr. Volker Riediger* für die gute Betreuung, die vielen hilfreichen Ratschläge und eine angenehme Arbeitsatmosphäre bedanken.

Ebenfalls möchte ich mich bei *Jens Wilhelmi* und *Dr. Ulrich Barjenbruch* von der *Bundesanstalt für Gewässerkunde* sowie bei den Mitarbeitern des Projektpraktikums „Simulation eines Pegel-Messnetzes“ *David Saile*, *Johann Raskatow*, *Alvaro Gauterin*, *Dmitriy Pichkurov*, *Heiko Richter*, *Marcel Becker*, *Michael Herold* und *Sascha Strauß* für die angenehme und gute Zusammenarbeit bedanken.

Weiterhin bedanke ich mich bei meinen Eltern, dass sie mich während meiner gesamten Studienzeit unterstützt und mir das Studium überhaupt erst ermöglicht haben. Meiner Freundin *Jennifer Bender* und *Volker Schäfer* danke ich für die ständige Motivation und das Korrekturlesen der Diplomarbeit.

Ein ganz besonderer Dank geht an alle, die mir in Zeiten schwerer Krankheit beistanden und die Kraft gegeben haben diese durchzustehen.

Inhaltsverzeichnis

1. Einleitung

Diese Diplomarbeit beschreibt die Entwicklung einer Web Service orientierten Umgebung zur Verarbeitung von Pegeldaten mehrerer Gewässer in Zusammenarbeit mit der Bundesanstalt für Gewässerkunde (BfG)[1]. Sie dokumentiert den gesamten Entwicklungsablauf und gibt eine Einführung in die verwendeten Technologien. Für *XHydro*[2], ein Austauschformat zum wirtschaftlichen und organisationsübergreifenden Austausch von Zeitreihen, wird zusätzlich eine Bewertung des Systems inklusive Verbesserungsvorschlägen gegeben.

1.1. Bundesanstalt für Gewässerkunde

Die *Bundesanstalt für Gewässerkunde* ist als wissenschaftliches Institut im Rang einer Bundesoberbehörde für die Bundeswasserstraßen (Fließ- und Küstengewässer sowie Kanäle) zuständig. Sie berät die Bundesministerien, sowie die Wasser- und Schifffahrtsverwaltung des Bundes (WSV)[3] in Bezug auf die Nutzung und Bewirtschaftung der Bundeswasserstraßen (siehe Abbildung 1.1).

Abbildung 1.1.: Einordnung der Bundesanstalt für Gewässerkunde [Bun05]

Um eine kompetente Beratung zu ermöglichen bedarf es mehrerer unterschiedlicher Betrachtungs- und Herangehensweisen. Zu diesem Zweck wurden die Kompetenzen in die drei Fachabteilungen *Quantitative Gewässerkunde*, *Qualitative Gewässerkunde* und *Ökologie* unterteilt, welche die unterschiedlichen Aspekte der Gewässerkunde betrachten.

Quantitative Gewässerkunde

Die Fachabteilung *Quantitative Gewässerkunde* beschäftigt sich mit der Untersu-

[1] `http://www.bafg.de` (abgerufen: 16.07.2011)
[2] `http://www.xhydro.de` (abgerufen: 16.07.2011)
[3] `http://www.wsv.de` (abgerufen: 16.07.2011)

chung der Wasserstände und Abflüsse, der Geometrie und dem morphologischen Zustand der Wasserstraßen. Die Untersuchungen gehen dabei weit über den engeren Bereich der Wasserstraßen hinaus und beschäftigen sich beispielsweise auch mit den Auen, dem flussnahen Grundwasser, den Entwicklungen im Einzugsgebiet der Wasserstraßen bis hin zu den Auswirkungen des globalen Klimawandels. Die Entwicklung von Messgeräten und -verfahren spielt dabei ebenso eine wichtige Rolle, wie die Qualitätssicherung der gewonnenen Daten. Durch diese lassen sich anhand von Simulations-, Prognose- und Vorhersagemodellen Aussagen über die Wirkung von wasserbaulichen und wasserwirtschaftlichen Maßnahmen sowie zur Abflussentwicklung im Einzugsgebiet machen.

Qualitative Gewässerkunde

Die Untersuchung der Schadstoffvorkommen in Gewässern ist das Kerngebiet der Fachabteilung *Qualitative Gewässerkunde*. Durch bundesweite Messungen der Wasserbeschaffenheit (insbesondere der Schwebstoffe und Sedimente) sowie durch projektbezogene Datenerhebungen lassen sich Wirkungsszenarien und Prognosen über die zukünftige Entwicklung erstellen. Ziel dieser Fachabteilung ist es, die negativen Auswirkungen der Unterhaltung und des Ausbaus von Bundeswasserstraßen auf die Gewässerbeschaffenheit zu minimieren.

Ökologie

Ökologische Betrachtungen in und an den Bundeswasserstraßen sind die Aufgabe der Fachabteilung *Ökologie*. Neben der Erforschung der Beschaffenheit von Gewässern, sowie dem Entwickeln von Konzepten und Maßnahmen für eine umweltverträgliche Bewirtschaftung der Wasserstraßen, werden auch Wege zur Minimierung negativer Auswirkungen durch menschliches Eingreifen in die Natur und die Nutzung der Wasserstraßen gesucht.

[Bun05]

Diese Diplomarbeit ist thematisch in der Fachabteilung *Quantitative Gewässerkunde*, genauer gesagt beim Referat M1 (Hydrometrie und gewässerkundliche Begutachtung), anzusiedeln (siehe Abbildung 1.2).

1.2. Motivation

Für die Binnenschifffahrt, wasserwirtschaftliche Maßnahmen des Bundes und der Länder, aber auch zur Information der Bevölkerung (z.B. bei Hochwasseralarm) sind die aktuellen Wasserstandsinformationen von großer Bedeutung. Die automatisierte Messung der Pegelstände an den Messstellen, sowie deren Übertragung an die Abrufzentralen spielt dabei eine wichtige Rolle.

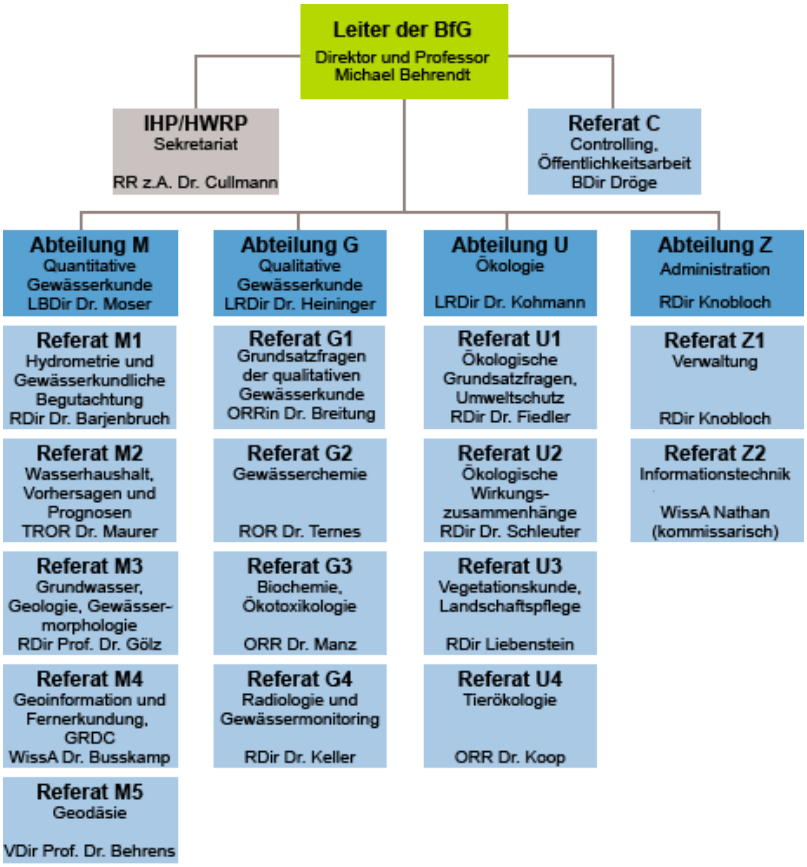


Abbildung 1.2.: Organisation der Bundesanstalt für Gewässerkunde [Bun05]

Da der Betrieb und die technische Ausstattung der Messstellen nicht einheitlich erfolgen kann, existiert eine Vielzahl an Protokollen und Formaten für den Datenaustausch, welche sich hauptsächlich an der Hardware und verfügbaren Leitungskapazität der jeweiligen Messstationen orientieren. Dies führt dazu dass die Abrufzentralen mit Multiportokollservern ausgestattet sein müssen, um die Daten trotz ihrer Verschiedenartigkeit zu bearbeiten. Das Referat M1 der Bundesanstalt für Gewässerkunde hat es sich zur Aufgabe gemacht, den Abruf der Pegelstände zu harmonisieren.

[Bun]

1.3. Ausgangslage

Die Wasserstände werden derzeitig alle 15 Minuten gemessen und lokal an den Messstellen zwischengespeichert bis sie durch das System SODA der Firma *Kisters*[4] abgerufen werden (derzeit dreimal täglich). Dieses schreibt die Daten in eine

[4] `http://www.kisters.de` (abgerufen: 16.07.2011)

von mehreren verteilten Datenbanken. Diese Daten lassen sich derzeitig durch das System *PEGELONLINE*[5] abrufen.

Die Pegeldaten durchlaufen weiterhin mehrere Workflows (siehe Abschnitt 1.4.1 bis Abschnitt 1.4.4), welche derzeitig manuell bearbeitet werden.

Zur Harmonisierung der Pegeldaten ließ die BfG von den Firmen *Kisters* und *disy*[6] den XML-Dialekt XHydro entwickeln [Bun].

Die Übertragung der XHydro-Daten geschieht durch Web Services. Hierbei wurde besonders auf Standardkonformität und Einhaltung der W3C-Spezifikationen geachtet. Dadurch soll es möglich sein, die Daten auch über diverse Zwischenstationen ohne Protokollkonvertierungen übertragen zu können [Bun08].

Vorbereitend wurde ein Test-Server unter der Adresse `http://www.xhydro-test.de` (abgerufen: 16.07.2011) eingerichtet, über welchen die Pegelstände von *Pegel Kaub*[7], sowie *Pegel Mainz* abgerufen werden können. Die Daten beider Pegel werden dort im XHydro-Format zur Verfügung gestellt. Der Pegel Kaub unterstützt bereits standardmäßig das XHydro-Format, während der Pegel Mainz nativ das Format iLON100 verwendet.

Weiterhin wurde von Florian Schricker im Rahmen seiner Diplomarbeit (siehe [Sch07]) das Programm *WaterViz* entwickelt, mit welchem sich die Daten des Pegel Mainz abrufen und visualisieren lassen. WaterViz wird evtl. zukünftig von der BfG weiterentwickelt.

Die weitere Planung der Bundesanstalt für Gewässerkunde sieht vor, XHydro in einem größeren Teilsystem zu implementieren. Darüber hinaus ist geplant, dass die Abrufzentralen die Daten nicht mehr abholen, sondern diese von den Messstellen automatisch über einen Push-Service verteilt werden. Dieser Push-Service ist bereits für die Pegel Kaub und Mainz implementiert.

1.4. Ziel der Diplomarbeit

Das Ziel dieser Diplomarbeit ist es, eine auf Standards basiersende Web Service orientierte Umgebung zur Verarbeitung der Pegeldaten zu entwickeln (im Folgenden *PegelSuite* genannt). Die wichtigsten verwendeten Standards sind XHydro und BPEL. PegelSuite muss dabei insbesondere skalierbar, performant und standardkonform sein. Bei der Entwicklung stehen 4 Workflows im Vordergrund. Diese werden in den folgenden Unterabschnitten genauer erläutert.

[5] `http://www.pegelonline.wsv.de` (abgerufen: 16.07.2011)
[6] `http://www.disy.net` (abgerufen: 16.07.2011)
[7] `http://www.pegel-kaub.de` (abgerufen: 16.07.2011)

1.4.1. Workflow 1: Datenabruf und -empfang

Der erste Workflow dient der Empfangskontrolle der Daten. Werden von einem Pegel eine bestimmte Zeit keine Daten empfangen (Push-Betrieb), so werden diese aktiv von der PegelSuite bei dem Pegel angefragt (Pull-Betrieb). Scheitert selbst der Abruf der Daten, wird ein Alarm gesendet (siehe Abschnitt 1.4.4). Wurden die Daten empfangen, so werden diese nach XHydro transformiert, falls sie in einem anderen Format (z.B. iLON100) vorliegen. Anschließend werden die Daten an den zweiten Workflow weitergereicht.

1.4.2. Workflow 2: Sensorvergleich

Im zweiten Workflow werden die einzelnen Sensoren des Pegels (sofern vorhanden) verglichen. Dafür wird der gleitende Mittelwert als Qualitätsmerkmal über alle Sensorwerte gebildet. Anschließend werden die einzelnen Sensorwerte zusammen mit dem Qualitätsmerkmal in der Datenbank gespeichert.

Sollte ein Sensor ausgefallen sein oder die Abweichung der Sensorwerte zum Qualitätsmerkmal zu groß sein, wird zusätzlich ein Alarm gesendet (siehe Abschnitt 1.4.4).

1.4.3. Workflow 3: Plausibilisierung

Die verglichenen Sensordaten aus Workflow 2 werden in regelmäßigen Abständen anhand eines gesamten Gewässerabschnitts plausibilisiert[8]. Für diesen Vorgang werden die Daten des am höchsten priorisierten Sensors verwendet.

Nach der eigentlichen Plausibilisierung wird anhand der plausibilisierten Daten die Qualität der Plausibilisierung berechnet. Diese wird zur Prüfung der Plausibilisierung benötigt. Wurde die Plausibilisierung erfolgreich abgeschlossen, so werden die plausibilisierten Daten in der Datenbank gespeichert. Sollte die Plausibilisierung fehlgeschlagen oder die Qualität nicht ausreichend sein, so wird eine Meldung an die entsprechenden Personen gesendet. Diese können die Plausibilisierung von Hand durchführen.

1.4.4. Workflow 4: Alarmmanangement

Bei der Ausführung der anderen Workflows kann es immer wieder dazu kommen, dass Alarmmeldungen an die entsprechenden Stellen gesendet werden müssen. Die Alarmierung kann beispielsweise per E-Mail oder in wichtigen Fällen evtl. per SMS erfolgen. In den folgenden Fällen wäre eine Alarmierung notwendig:

[8] Auf die eigentliche Plausibilisierung wird nicht weiter eingegangen, da sie nicht Teil der Diplomarbeit ist. Stattdessen wird eine prototypische Fuzzylogik von der BfG bereitgestellt.

- Workflow 1: Datenabruf und -empfang
  - Abruf der Daten x-mal gescheitert
- Workflow 2: Sensorvergleich
  - Sensoren fehlerhaft
  - Abweichung zu groß
- Workflow 3: Plausibilisierung
  - Abweichung zu groß
  - Fehlende Daten

Beim Auftreten eines Fehlers wird die erste Person aus einer Liste alarmiert. Wurde der Alarm nach einer bestimmten Zeit nicht bearbeitet, wird die nächste Person der Liste alarmiert (nächste Eskalationsstufe). Ob ein Alarm bereits von einer Person bearbeitet wird, kann durch eine erfolgte Quittierung nachgeprüft werden.

1.4.5. Skalierbarkeit, Performanz und Standardkonformität

Wie bereits erwähnt soll die PegelSuite skalierbar, performant und standardkonform sein. Die *Skalierbarkeit* und die *Performanz* sind notwendig, da das System später rund 1450 Pegel umfassen soll, welche alle 15 Minuten ihre Daten an die PegelSuite senden.

Die *Standardkonformität* wird durch Verwendung der Standards des W3C, OASIS etc. erreicht. Von der Bundesanstalt für Gewässerkunde wurde dabei *WS-BPEL* (siehe Abschnitt 2.4) als Standard für die Orchestrierung der Web Services vorgegeben. Für den Datenaustausch wird der von der BfG eingeführte Standard *XHydro* (siehe Abschnitt 2.5) verwendet.

1.4.6. Zusammenfassung

PegelSuite soll somit, auf Standards für Web Services basierend, Pegeldaten deutscher Gewässer verarbeiten. Diese werden dafür von den Pegeln an die PegelSuite gesendet oder von dieser abgerufen. Diese Daten werden anschließend auf Fehler wie z.B. Minuswerte überprüft. In regelmäßigen Abständen werden die Daten eines gesamten Gewässerabschnitts plausibilisiert. Sollten bei der Verarbeitung Fehler oder zu große Abweichungen festgestellt werden, wird ein Alarm an die zuständigen Personen gesendet.

2. Technologien

Für die Implementierung der PegelSuite sind eine Vielzahl von Technologien notwendig. Zum besseren Verständnis der späteren Kapitel werden diese Technologien in den folgenden Abschnitten genauer beschrieben.

2.1. Exkurs: XML

Da für viele der folgenden Kapitel Vorkenntnisse in XML benötigt werden, wird diese Sprache in diesem Exkurs kurz eingeführt.

XML (Extensible Markup Language) ist eine *Auszeichnungssprache*, welche *hierarchisch strukturierte Daten* und teilweise auch *das Verhalten der verarbeitenden Programme beschreibt*.

Sie wurde von dem *World Wide Web Consortium (W3C)*[1] aus einer Teilmenge der Sprache SGML spezifiziert. Das Ziel war es, einen Standard zu schaffen, welcher sich im Internet auf einfache Weise nutzen lässt und ein breites Spektrum von Anwendungen unterstützen soll. Dabei soll XML *einfach*, *verständlich* und *menschenlesbar* sein.

Physische Struktur

Der physische Aufbau eines XML-Dokuments besteht aus Einheiten, welche einen Inhalt haben und durch einen Namen identifiziert werden können (Entitäten). Diese können wiederum andere Entitäten referenzieren. Eine Entität besteht aus einer Zeichenfolge, welche Auszeichnungdaten oder Textdaten repräsentieren kann.

Logische Struktur

Neben der physischen Struktur besitzt ein XML-Dokument auch eine logische Struktur. Diese entspricht einer Baumstrukur und besteht aus *Elementen*, *Attributen*, *Kommentaren*, *Verarbeitungsanweisungen* und *Textdaten*. Ein *Element* ist ein von Tags umgebenes XML-Konstrukt. In einem XML-Dokument existiert genau ein Hauptelement, das sogenannte *Wurzelelement*. Innerhalb des Wurzelelements können weitere Elemente beliebig ineinander geschachtelt werden. *Attribute* sind im Start-Tag eines Elements enthalten und enthalten zusätzliche Informationen zum Element. Um Elemente für den Menschen genauer zu beschreiben, ohne

[1] `http://www.w3.org` (abgerufen: 16.07.2011)

dass diese Informationen bei der Verarbeitung des Dokuments verwendet werden, nutzt man *Kommentare*. Die *Verarbeitungsanweisungen* werden verwendet, um Anweisungen an Anwendungen in XML einzubetten. Die *Textdaten* bestehen aus allen Daten, welche keine Auszeichnungsdaten sind (z.B. Elementwerte oder Attributwerte).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Describes the water gauge data, including location,
3  time and value -->
4 <watergaugedata>
5  <location>Kaub</location>
6  <time>2009-12-27T17:15:00+01:00</time>
7  <value unit="cm">315.5</value>
8 </watergaugedata>
```

Listing 2.1: XML-Beispiel

In dem Beispiel in Listing 2.1 handelt es sich in Zeile 1 um eine Verarbeitungsanweisung, welche in diesem Fall die verwendete XML-Version sowie die Zeichenkodierung angibt. In den Zeilen 2 und 3 steht ein Kommentar, welcher das Dokument näher beschreibt. Zeile 4 bis 8 enthält das Wurzelelement `watergaugedata` es enthält alle weiteren Elemente in den Zeilen 5 - 7. Das Element `value` in Zeile 7 beschreibt den Pegelstand anhand seines Wertes. Weiterhin wird durch das Attribut `unit` die zugehörige Einheit angegeben. Textdaten wären bei diesem Element `cm` und `315.5`.

Dieses XML-Dokument beschreibt[2] also einen Pegelstand von 315.5 cm am 27.12.2009 um 17:15:00 (MEZ) am Pegel Kaub.

[W3C08a]

2.1.1. Wohlgeformtheit und Validität

Ein korrektes XML-Dokument muss sowohl *wohlgeformt* als auch *valide* (gültig) sein.

Ein XML-Dokument ist wohlgeformt, wenn es die folgenden Kriterien erfüllt:

- Das Dokument besitzt genau ein Wurzelelement, dass alle anderen Elemente umgibt.
- Jeder geöffnete Tag muss auch wieder geschlossen werden.

[2] Dieses Beispiel dient lediglich zur Veranschaulichung. In Abschnitt 2.5 wird mit XHydro eine konkrete Sprache zur Beschreibung von Zeitreihen vorgestellt.

- Die Elemente müssen in der korrekten Reihenfolge (z.B. `<a><b></b></a>` und nicht `<a><b></a></b>`) wieder geschlossen werden.

Als valide wird ein XML-Dokument bezeichnet, wenn es sich an eine angegebene Grammatik - wie eine XML-Schema-Beschreibung (siehe Abschnitt 2.1.3) - hält.

[W3C08a]

2.1.2. Namensräume

Durch eine Grammatik lassen sich XML-Dokumente einschränken. Dadurch entstehen verschiedene Sprachen in XML, wie beispielsweise WSDL, SOAP, BPEL oder XHydro. Um verschiedene dieser Sprachen innerhalb eines Dokuments verwenden zu können, wurden in XML Namensräume eingeführt. Wenn sich beispielsweise am Pegel Kaub auch eine Wetterstation befinden würde, so könnte man die Temperatur auch zusammen mit dem Wasserstand abrufen, selbst wenn hierfür eine andere XML-Sprache - speziell für Wetterdaten - verwendet würde.

In Listing 2.2 ist solch eine Sprache vorausgesetzt. In den Zeilen 2 und 3 werden die Namensräume durch `xmlns`-Attribute angegeben. Zeile 2 definiert den Standard-Namensraum des Dokuments und Zeile 3 gibt eine Referenz auf den fremden Namensraum an. Die Namensräume werden immer anhand eines URI definiert. Dieser sollte einzigartig und dauerhaft sein, muss jedoch nicht zwingend existieren.

Vergleicht man Zeile 6 und 7, so stellt man fest, dass beide Sprachen ein Element `value` besitzen. Diese Elemente gehören jedoch verschiedenen Namensräumen an, was durch den Präfix `w:` ausgedrückt wird.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <watergaugedata xmlns="http://www.pegel-kaub.de/watergaugedata"
3  xmlns:w="http://www.pegel-kaub.de/weather">
4  <location>Kaub</location>
5  <time>2009-12-27T17:15:00+01:00</time>
6  <value unit="cm">315.5</value>
7  <w:value unit="degree_centigrade">0.5</w:value>
8 </watergaugedata>
```

Listing 2.2: XML-Namensraum-Beispiel

Auf diese Art und Weise lassen sich innerhalb eines XML-Dokuments beliebig viele XML-Sprachen verwenden.

[W3C09a]

2.1.3. XML Schema

Um eine XML-Sprache zu definieren muss eine Grammatik erstellt werden. Dies geschieht normalerweise durch XML Schema, welches ebenfalls eine XML-Sprache[3] ist. Anhand des erstellten Schemas können XML-Parser die Validität der dazugehörigen XML-Dokumente prüfen.

Eine XML Schema Definition (XSD) beschreibt neben der Struktur des XML-Dokuments auch die Datentypen der jeweiligen Elemente (`element`) oder Attribute (`attribute`).

Bei den Datentypen wird zwischen einfachen und komplexen Datentypen unterschieden. Die *einfachen Datentypen* sind die aus den meisten Programmiersprachen bekannten Typen wie etwa `string` oder `float`. Die *komplexen Datentypen* (`complexType`) dagegen können weitere Elemente und Attribute enthalten [W3C04c].

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
5   <xsd:element name="watergaugedata">
6    <xsd:complexType>
7     <xsd:sequence>
8      <xsd:element name="location" type="xsd:string" />
9      <xsd:element name="time" type="xsd:dateTime" />
10     <xsd:element name="value">
11      <xsd:complexType>
12       <xsd:simpleContent>
13        <xsd:extension base="xsd:float">
14         <xsd:attribute name="unit" type="xsd:string" />
15        </xsd:extension>
16       </xsd:simpleContent>
17      </xsd:complexType>
18     </xsd:element>
19    </xsd:sequence>
20   </xsd:complexType>
21  </xsd:element>
22  </xsd:schema>
```

Listing 2.3: XML Schema Beispiel

Listing 2.3 zeigt die XML Schema Definition der Sprache des in Listing 2.1 vorgestellten Beispiels. Das Wurzelelement `watergaugedata` (Zeilen 5 - 21) definiert einen komplexen Typ (Zeile 6 - 20), welcher die Elemente `location`, `time`

[3] Die Grammatik von XML Schema kann wiederum durch XML Schema beschrieben werden.

und `value` (Zeilen 8-18) enthält. `location` (Zeile 8) und `time` (Zeile 9) werden als Elemente definiert, welche einen einfachen Datentyp enthalten. Das Element `value` (Zeilen 10 - 18) dagegen besteht aus einen komplexen Typ (Zeilen 11 - 17), der in den Zeilen 13 - 15 um den Datentyp des Elements (Zeile 13) und das Attribut `unit` vom Datentyp `string` (Zeile 14) erweitert wird.

2.2. Exkurs: Serviceorientierte Architektur

Um Web Services (Abschnitt 2.3) besser zu verstehen, ist es hilfreich, vorher etwas über Serviceorientierte Architektur (SOA) zu erfahren, da es sich bei Web Services um eine spezielle Realisierung einer SOA handelt.

Der Begriff *Serviceorientierte Architektur* beschreibt ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet werden kann [OAS06].

Das Kernelement einer SOA ist der *Service*. Ein Service ist eine Leistung, die von einem Dritten über eine vorgeschriebene Schnittstelle angeboten und erbracht wird und sich an die dort angegebenen Beschränkungen und Richtlinien hält [OAS06].

2.2.1. Prinzipien einer SOA

Serviceorientierte Architekturen folgen einigen wesentlichen Prinzipien. Die wichtigsten dabei sind:

- Lose Kopplung
- Abkapselung
- Verteilung
- Wiederverwendbarkeit
- Prozessorientiertheit
- Komponierbarkeit
- Unabhängigkeit
- Zustandslosigkeit
- Interoperabilität
- Auffindbarkeit

Durch das Prinzip *Lose Kopplung* wird die Abhängigkeit zwischen zwei Services gering gehalten. Dadurch ist es möglich, die interne Logik eines Services, mit keinen oder minimalen Auswirkungen auf die anderen Services, zu ändern.

Aus diesem Grunde werden auch nur für die Nutzung relevante Informationen der Schnittstellenbeschreibung hinzugefügt (*Abkapselung*). Über diese erhalten

die Nutzer des Services alle Informationen, die zur Verwendung des Dienstes erforderlich sind.

Kleine Services machen es ebenfalls möglich, diese global zu verteilen (*Verteilung*). Meist sind an einem Geschäftsprozess mehrere Unternehmen beteiligt, welche jeweils ihre Dienste anbieten. Diese sind, über ein Netzwerk zugreifbar, global verteilt.

Die *Wiederverwendbarkeit* eines Services ist nötig, um diesen für potentiell mehrere Servicenehmer zugänglich zu machen.

Um möglichst unabhängig zu sein, sollte ein Service möglichst klein und auf einen Prozess beschränkt sein (*Prozessorientiertheit*).

Durch das Prinzip der *Komponierbarkeit* lassen sich kleinere Services wieder zu einem großen Ganzen vereinen. Dies kann beispielsweise für Web Services durch WS-BPEL (siehe Abschnitt 2.4) geschehen.

Wichtig ist auch die *Unabhängigkeit* der Services, d.h. sie verwenden nur ihre eigene Logik. Durch diese lassen sich die Prinzipien Lose Kopplung, Verteilung, Wiederverwendbarkeit und Komponierbarkeit leichter erreichen.

Durch das Prinzip der *Zustandslosigkeit* wird sichergestellt, dass Services alle Anfragen als eigenständige Transaktionen behandeln und ihren Zustand nicht anhand vorangegangener Aktionen beibehalten. Jede Servicenutzung wird als komplett eigenständige Anfrage behandelt. Dadurch lassen sich die Prinzipien Lose Kopplung, Wiederverwendbarkeit und Kombinierbarkeit leichter erfüllen.

Die *Interoperabilität* von Diensten wird dadurch gewährleistet, dass diese nur über ihre öffentlichen Schnittstellen kommunizieren und sich dabei an bestehende Standards halten.

Ein weiterer wichtiger Aspekt einer Serviceorientierten Architektur ist die *Auffindbarkeit* eines Services. Um diesen zu nutzen, muss seine Schnittstellenbeschreibung an einem öffentlich zugänglichen Ort hinterlegt sein.

[Vas07]

2.3. Web Services

Web Services sind die derzeitig populärste Realisierung einer Serviceorientierten Architektur. Bei einem Web Service handelt es sich um eine Softwarekomponente, welche eine Machine-to-Machine (M2M) Interaktion mittels auf XML basierender Sprachen über ein Netzwerk ermöglicht [W3C04a].

Es arbeiten dabei mehrere Akteure zusammen (siehe Abbildung 2.1): *Requester*, *Provider* und *Broker* [Gun02].

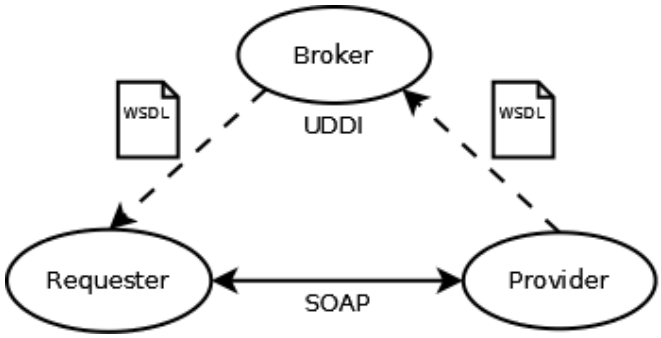


Abbildung 2.1.: Das „Web Service Dreieck“ [Gun02]

Der *Broker* stellt einen Verzeichnisdienst für Web Services (UDDI) zur Verfügung, in welchem der *Provider* seinen Web Service veröffentlichen kann. Dies geschieht durch eine WSDL-Datei, welche den Service beschreibt. Diese WSDL-Datei kann der *Requester* herunterladen und mithilfe der darin enthalten Informationen SOAP-Nachrichten mit dem Provider austauschen.

Die Funktionsweise eines Web Services lässt sich in 4 Aspekte unterteilen [W3C04a]:

- Beschreibung (WSDL)
- Implementierung
- Publizierung (UDDI)
- Kommunikation (SOAP)

In den folgenden Unterabschnitten werden diese Aspekte genauer beschrieben.

2.3.1. WSDL

Um einen Web Service zu beschreiben wird die auf XML basierende Sprache WSDL (Web Services Description Language) verwendet. Sie beschreibt die Nachrichten, welche zwischen Requester und Provider ausgetauscht werden können. Die Nachrichten selbst werden dabei abstrakt beschrieben und danach an ein konkretes Protokoll und Nachrichtenformat gebunden [W3C04a].

Struktur einer WSDL-Datei

Eine WSDL-Datei enthält die 5 Grundelemente `types`, `message`, `portType`, `binding` und `service` [Gun02]. Diese Elemente entsprechen den orangefarbenen Elementen in Abbildung 2.2. Diese zeigt die Struktur einer WSDL-Datei, welche sich in einen inhaltlichen Teil und einen technischen Teil zerlegen lässt. Der inhaltliche Teil beschreibt den Inhalt der Schnittstelle und der technische Teil die Bindung an Protokolle und Adressen.

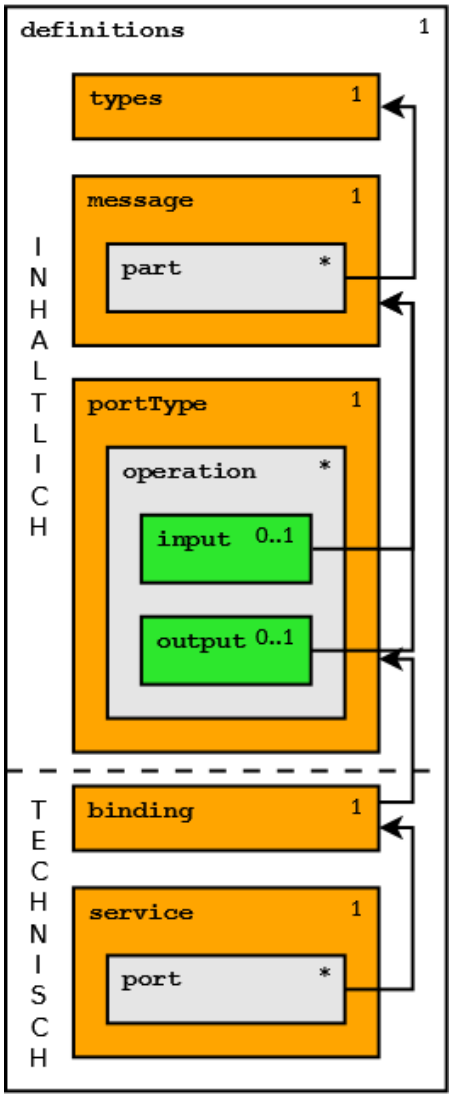


Abbildung 2.2.: Struktur einer WSDL-Datei

Diese Elemente, welche im Folgenden genauer beschrieben werden, referenzieren sich über Attribute gegenseitig (Pfeile in Abbildung 2.2). Das Kindelement `port` von `service` referenziert beispielsweise `binding`.

Bei der Referenzierung einer Nachricht (`message`) wird zudem unterschieden, ob es sich um eine eingehende (`input`) oder ausgehende (`output`) Nachricht handelt. Dementsprechend wird die Nachricht entweder von `input` oder `output` referenziert.

Anhang A.1 zeigt ein fiktives Beispiel einer WSDL-Datei. In den folgenden Unterabschnitten werden Ausschnitte aus diesem Beispiel genauer erläutert.

Das Wurzelelement `definitions`

`definitions` ist das Wurzelelement einer WSDL-Datei und es enthält die 5 Grundelemente.

Listing 2.4 zeigt dieses Element einer jeden WSDL-Datei. Dort wird der Name (Zeile 2) der WSDL-Beschreibung sowie die verwendeten Namensräume (Zeilen 3 - 7) als Attribut angeben. Der eigene Namensraum der WSDL-Datei wird in Zeile 3 definiert und in Zeile 4 mit dem Präfix `tns` versehen. Der Namensraum in Zeile 5 wird für die Schema-Beschreibung der WSDL-Datei verwendet. Die weiteren Namensräume werden für SOAP- (Zeile 6) und WSDL-Sprachelemente (Zeile 7) benötigt. *Beispiel*

```
2 <definitions name="WaterGaugeData"
3  targetNamespace="http://www.pegel-kaub.de/watergaugedata.wsdl"
4  xmlns:tns="http://www.pegel-kaub.de/watergaugedata.wsdl"
5  xmlns:xsd1="http://www.pegel-kaub.de/watergaugedata.xsd"
6  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Listing 2.4: WSDL-Beispiel: `definitions`

Types

Das `types`-Element enthält Datentyp-Definitionen, welche für den Austausch von Nachrichten zwischen Provider und Requester notwendig sind. Dies geschieht üblicherweise durch eine XML Schema Definition. Dies hat den Vorteil eines hohen Maßes an Interoperabilität und Plattformunabhängigkeit.

Für die Nutzung von XSD gelten jedoch bei WSDL-Beschreibungen spezielle Regeln:

- Es dürfen nur Elemente und keine Attribute definiert werden.
- Attribute oder Elemente, welche technische Aspekte beschreiben, dürfen nicht verwendet werden.
- Array-Datentypen sollten den Array-Datentyp des „SOAP 1.1 Kodierungsschemas“[4] erweitern, ungeachtet dessen, ob die Kodierung (siehe Kapitel 5 in [W3C00]) letztendlich verwendet wird. Der Name des Arrays sollte dabei `ArrayOf`*`Type`* sein, wobei *`Type`* der Datentyp der Elemente des Arrays ist. Um diese, sowie die Array-Dimensionen, zu spezifizieren wird der Standardwert des Attributs `arrayType` des „SOAP 1.1 Kodierungsschemas“ verwendet.

[4] Namensraum: `http://schemas.xmlsoap.org/soap/encoding`

- Um Felder oder Parameter zu beschreiben, welche irgendeinen Datentyp annehmen können, muss der Datentyp `anyType` von XML Schema verwendet werden.

[W3C01]

Beispiel In Listing 2.5 ist ein Beispiel für `types` angegeben. In Zeile 11 wird der eigene Namensraum der Schema-Beschreibung definiert, dieser entspricht dem aus Zeile 5 in Listing 2.4. Weiterhin werden die Typen `Header` (Zeilen 13 - 19), `WaterGaugeDataRequest` (Zeilen 20 - 26) und `WaterGaugeData` (Zeilen 27 - 34) definiert.

Der Typ `Header` definiert ein Element `Duration` vom Typ `integer` (Zeile 16). `Duration` wird später im Header einer SOAP-Nachricht (siehe Abschnitt 2.3.4 verwendet um die Verarbeitungsdauer des Aufrufs in Millisekunden angeben.

`WaterGaugeDataRequest` definiert das Element `time` vom Typ `dateTime` in Zeile 23. So kann durch Angabe der Zeit, der Pegelstand zu diesem Zeitpunkt abgerufen werden.

Der Pegelstand (`WaterGaugeData`) wird durch Angabe des Wertes (`value`; Zeile 30) und der zugehörigen Einheit (`unit`; Zeile 31) angegeben. `unit` wurde hier nicht als Attribut von `value` definiert (wie etwa in Listing 2.2), sondern als eigenständiges Element. Dies liegt an der Einschränkung, dass bei XML Schema in einer WSDL-Datei keine Attribute erlaubt sind.

```
9  <types>
10  <schema
11   targetNamespace="http://www.pegel-kaub.de/watergaugedata.xsd"
12   xmlns="http://www.w3.org/2001/XMLSchema">
13   <element name="Header">
14    <complexType>
15     <all>
16      <element name="Duration" type="integer" />
17     </all>
18    </complexType>
19   </element>
20   <element name="WaterGaugeDataRequest">
21    <complexType>
22     <all>
23      <element name="time" type="dateTime" />
24     </all>
25    </complexType>
26   </element>
27   <element name="WaterGaugeData">
```

```
28    <complexType>
29     <all>
30      <element name="value" type="float" />
31      <element name="unit" type="string" />
32     </all>
33    </complexType>
34   </element>
35  </schema>
36 </types>
```

Listing 2.5: WSDL-Beispiel: `types`

Messages

Das Element `message` enthält eine abstrakte, typisierte Definition der Parameter einer Nachricht. Es besteht aus einem oder mehreren logischen Teilen (`part`). Jeder Part verweist auf einen Datentyp, welcher in `types` definiert wurde. Dies geschieht bei Verwendung von XML Schema Definitionen entweder durch das Attribut `element` oder durch das Attribut `type`. `element` bezieht sich dabei auf ein Element, während `type` sich auf einen einfachen (`simpleType`) oder komplexen Typen (`complexType`) bezieht.

part-Element

Durch das `part`-Element werden die Parameter einer Nachricht festgelegt. Normalerweise enthält ein `part` genau einen Parameter, jedoch kann dieser durch eine Schema Definition in `types` weitere Kindelemente (also weitere Parameter) enthalten.

[W3C01]

Beispiel

In Listing 2.6 ist ein Beispiel für das Element `message` angegeben. Die bereits in Listing 2.5 definierten Datentypen werden nun zu Nachrichten gruppiert. In den Zeilen 38 - 40 wird die Nachricht `Header` definiert. Diese ist vom Typ `Header` (Zeile 39). `GetWaterGaugeDataInput` (Zeilen 41 - 43) enthält einen Parameter vom Typ `WaterGaugeDataRequest` (Zeile 42), welcher in `types` definiert wurde. `GetWaterGaugeDataOutput` in den Zeilen 44 - 46 enthält einen Parameter vom Typ `WaterGaugeData` (Zeile 45).

```
38 <message name="Header">
39  <part name="header" element="xsd1:Header" />
40 </message>
41 <message name="GetWaterGaugeDataInput">
42  <part name="body" element="xsd1:WaterGaugeDataRequest" />
43 </message>
44 <message name="GetWaterGaugeDataOutput">
```

```
45  <part name="body" element="xsd1:WaterGaugeData" />
46 </message>
```

Listing 2.6: WSDL-Beispiel: `message`

PortTypes

Die abstrakte Beschreibung der Operationen eines Web Services wird durch das Element `portType` angegeben. Zu diesem Zweck werden die Operationen durch das Kindelement `operation` definiert. Diese bestehen aus Eingabedaten (`input`), Ausgabedaten (`output`), sowie Fehlern (`fault`). Der Inhalt der Nachricht wird hierbei durch das Attribut `message` referenziert.

[W3C01]

Beispiel Listing 2.7 beschreibt, wie die Nachrichten aus Listing 2.6 dem PortType `WaterGaugeDataPortType` (Zeilen 48 - 53), als Eingabe- bzw. Ausgabedaten der Operation `GetWaterGaugeData` (Zeilen 49 - 52) hinzugefügt werden. In diesem Fall wurde in Zeile 50 die Nachricht `GetWaterGaugeDataInput` als Eingabe- und `GetWaterGaugeDataOutput` in Zeile 51 als Ausgabeparameter verwendet.

```
48 <portType name="WaterGaugeDataPortType">
49  <operation name="GetWaterGaugeData">
50   <input message="tns:GetWaterGaugeDataInput" />
51   <output message="tns:GetWaterGaugeDataOutput" />
52  </operation>
53 </portType>
```

Listing 2.7: WSDL-Beispiel: `portType`

Bindings

Das `binding`-Element ist das erste technische Element einer WSDL-Datei und weist einem PortType ein konkretes Protokoll (meist SOAP) und Datenformat zu. Ein Binding beschreibt dabei genau ein Protokoll, aber es können auch mehrere `binding`-Elemente zu einem PortType existieren. Die Bindung des Protokolls und des Datenformats geschieht durch Erweiterungselemente des jeweiligen Protokolls. Für diesen Zweck werden die Nachfahren von `portType` wiederholt und an den entsprechenden Stellen die Erweiterungselemente eingefügt. Diese beziehen sich immer auf das Elternelement.

[W3C01]

In Listing 2.8 ist ein Beispiel für das `binding`-Element angegeben. Durch die Angabe des Attributs `type` (Zeile 56) wird der zugehörige PortType angegeben. *Beispiel*

Die konkreten Beschreibungen des Bindings werden durch die Erweiterungselemente des jeweiligen Protokolls angegeben. Dies geschieht in Listing 2.8 für SOAP. Zeile 57 gibt den Style des Aufrufs an. In diesem Fall wird `document` verwendet. Das bedeutet, dass der Aufruf ein Dokument enthält. Dass der Transport über HTTP geschieht ist in Zeile 58 definiert. Das Attribut `soapAction` in Zeile 61 spezifiziert den Wert des SOAPAction-Headers (siehe Abschnitt 2.3.4) für diese Operation und die Zeilen 63 und 68 geben durch das Attribut `use` an, ob eine Kodierung verwendet wird oder ein konkretes XML Schema (`literal`) vorliegt. Weiterhin werden diese Nachrichtenteile für den SOAP-Body definiert. In den Zeilen 66 und 67 wird für die ausgehende Nachricht zusätzlich ein SOAP-Header definiert. Er enthält das Element `Duration`, welches in Listing 2.5 definiert wurde.

```
55 <binding name="WaterGaugeDataSoapBinding"
56  type="tns:WaterGaugeDataPortType">
57  <soap:binding style="document"
58   transport="http://schemas.xmlsoap.org/soap/http" />
59  <operation name="GetWaterGaugeData">
60   <soap:operation
61    soapAction="http://www.pegel-kaub.de/getwatergaugedata" />
62   <input>
63    <soap:body use="literal" />
64   </input>
65   <output>
66    <soap:header message="tns:Header" part="Duration"
67     use="literal" />
68    <soap:body use="literal" />
69   </output>
70  </operation>
71 </binding>
```

Listing 2.8: WSDL-Beispiel: `binding`

Services

Ein `service`-Element gruppiert die Endpunkte (`port`) eines Web Services. Ein *Endpunkt* beschreibt eine spezifische Adresse für den Zugriff auf einen Service. Dies geschieht durch die Angabe eines spezifischen Protokolls und Datenformats.

Durch das `port`-Element wird der Service an eine spezifische Adresse und ein Binding gebunden. Jeder Port bindet dabei genau eine Adresse an ein Binding.

Dies geschieht durch ein Erweiterungselement.

[W3C01]

Beispiel Listing 2.9 beschreibt wie Bindings an einen Endpunkt gebunden werden. Dies geschieht durch das Element `service`. Dessen Kindelement `port` (Zeilen 75 - 79) enthält dabei die Referenz eines Bindings (Zeile 76), sowie die Adresse des Web Services (Zeilen 77, 78), welche durch das Attribut `location` des SOAP-Erweiterungselement `address` definiert wird.

In Zeile 74 wird das Element `documentation` verwendet. Durch dieses Element kann eine menschenlesbare Dokumentation angegeben werden. Das `documentation`-Element kann innerhalb jedes WSDL-Elements vorkommen. [W3C01]

```
73 <service name="WaterGaugeDataService">
74  <documentation>My first service</documentation>
75  <port name="WaterGaugeDataPort"
76   binding="tns:WaterGaugeDataSoapBinding">
77   <soap:address
78    location="http://www.pegel-kaub.de/watergaugedata" />
79  </port>
80 </service>
```

Listing 2.9: WSDL-Beispiel: `service`

2.3.2. Implementierung

Die Implementierung eines Web Services kann in jeder Programmiersprache, welche die nötigen Funktionen bereitstellt (z.B. Java, C++, PHP), realisiert werden. Die Implementierung wird in dieser Diplomarbeit nicht weiter erläutert.

2.3.3. Publizierung

Als Standard für die Publizierung eines Web Services dient der Verzeichnisdienst UDDI (Universal Description, Discovery and Integration). Dieser wird in der Praxis jedoch selten verwendet. IBM zum Beispiel ist der Meinung, dass es Zeit für einen neuen Standard wird, da UDDI bereits an seine Grenzen stößt (siehe [McK07]).

Für die Diplomarbeit ist dieser Dienst ebenfalls nicht von Nöten und wird daher nicht weiter erläutert.

2.3.4. SOAP

Um Nachrichten zwischen Provider und Requester, gemäß der mittels WSDL definierten Schnittstellen (siehe Abschnitt 2.3.1), zu verschicken wird SOAP verwendet. Entgegen einer weit verbreiteten Darstellung (siehe [W3C00]) handelt es sich bei SOAP nicht um ein Protokoll, sondern um eine auf XML basierende Sprache. Das verwendete Protokoll zur Übertragung ist in den meisten Fällen HTTP.

Struktur einer SOAP-Nachricht

Eine SOAP-Nachricht enthält die 2 Grundelemente `Header` und `Body`, welche vom Wurzelelement `Envelope` umschlossen werden (siehe Abbildung 2.3).

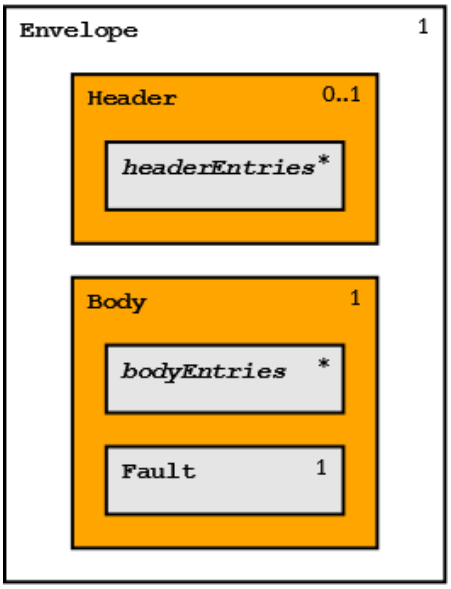


Abbildung 2.3.: Struktur einer SOAP-Nachricht

[W3C00]

Ein einführendes Beispiel

Beispiele für SOAP-Nachrichten sind Listing 2.10 und Listing 2.11. Bei Listing 2.10 handelt es sich um einen HTTP Request an einen Web Service und bei Listing 2.11 um die entsprechende HTTP Response. Die SOAP-Nachrichten legen die WSDL-Datei aus Anhang A.1 zugrunde.

```
1 POST /WaterGaugeData HTTP/1.1
2 Host: www.pegel-kaub.de
3 Content-Type: text/xml; charset="utf-8"
4 Content-Length: 364
5 SOAPAction: "http://www.pegel-kaub.de/getwatergaugedata"
6
```

```
7  <SOAP-ENV:Envelope
8   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
9   SOAP-ENV:encodingStyle=
10   "http://schemas.xmlsoap.org/soap/encoding/">
11  <SOAP-ENV:Body>
12   <m:GetWaterGaugeData
13    xmlns:m="http://www.pegel-kaub.de/watergaugedata.wsdl">
14    <m:time>2010-01-26T17:53:00+01:00</m:time>
15   </m:GetWaterGaugeData>
16  </SOAP-ENV:Body>
17 </SOAP-ENV:Envelope>
```

Listing 2.10: SOAP-Beispiel: HTTP Request

HTTP Request

Der HTTP Request ist in Listing 2.10 dargestellt. Zeile 1 - 5 gibt den HTTP-Header an, wobei Zeile 5 den Zweck des SOAP-Request angibt [W3C00]. Die SOAP-Nachricht selbst folgt im Body des HTTP Requests (Zeilen 7 - 17). Zeile 8 gibt dabei den Namensraum von SOAP an und Zeilen 9 und 10 geben das Kodierungschema der Nachricht an. In dieser SOAP-Nachricht existiert kein Header sondern lediglich ein Body (Zeilen 11 - 16). Das Element `GetWaterGaugeData` (Zeilen 12 - 15) definiert in Zeile 13 den Namensraum dieses Elements und Zeile 14 gibt die Zeit an, für welche der Wert abgefragt werden soll.

```
1  HTTP/1.1 200 OK
2  Content-Type: text/xml; charset="utf-8"
3  Content-Length: 390
4
5  <SOAP-ENV:Envelope
6   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7   xmlns:m="http://www.pegel-kaub.de/watergaugedata.wsdl"
8   SOAP-ENV:encodingStyle=
9    "http://schemas.xmlsoap.org/soap/encoding/">
10  <SOAP-ENV:Header>
11   <m:Duration>8</m:Duration>
12  </SOAP-ENV:Header>
13  <SOAP-ENV:Body>
14   <m:GetWaterGaugeDataResponse
15    xmlns:m="http://www.pegel-kaub.de/watergaugedata.wsdl">
16    <m:value>169.5</m:value>
17    <m:unit>cm</m:unit>
18   </m:GetWaterGaugeDataResponse>
19  </SOAP-ENV:Body>
20 </SOAP-ENV:Envelope>
```

Listing 2.11: SOAP-Beispiel: HTTP Response

HTTP Response

Die Antwort in Listing 2.11 ist vom Aufbau der des Requests sehr ähnlich. Lediglich der Header (Zeilen 10 - 12) und der Body (Zeilen 13 - 19) sind interessant. Im Header wird durch das Element `Duration` die Verarbeitungsdauer des Aufrufs in Millisekunden angegeben. Bei dem Element `GetWaterGaugeDataResponse` (Zeilen 14 - 18) handelt es sich um das Element `GetWaterGaugeData`, jedoch wird bei einer SOAP Antwort dass Suffix *Response* an den Elementnamen angefügt. In Zeile 16 wird der Pegelstand angegeben und Zeile 17 gibt die zugehörige Einheit an.

Das Wurzelelement `Envelope`

`Envelope` ist das Wurzelelement einer SOAP-Nachricht. Dieses enthält Informationen über den Namensraum von SOAP (`http://schemas.xmlsoap.org/soap/envelope`) und über das verwendete Kodierungsschema (normalerweise `http://schemas.xmlsoap.org/soap/encoding`).

SOAP Kodierungsschema

Das SOAP Kodierungsschema gibt die Serialisierungsregeln der SOAP-Nachricht an. Dies geschieht durch das Attribut `encodingStyle`. Dieses Attribut lässt sich für jedes Element angeben und gilt für alle Elemente und Kindelemente, sofern diese kein eigenes Kodierungsschema angeben.

[W3C00]

Header

Im optionalen `Header`-Element werden die Meta-Informationen der Nachricht angegeben. Dies können beispielsweise Authentifizierungsinformationen oder Informationen zur Kodierung der Daten sein. Existiert ein Header in einer SOAP-Nachricht, so muss dieser das erste Element nach dem Envelope sein. In bestimmten Fällen können auch mehrere Header innerhalb einer SOAP-Nachricht existieren.

Das `Header`-Element enthält außerdem zwei Attribute, die Informationen darüber enthalten, wie ein Empfänger die Nachricht verarbeiten soll:

`actor`-Attribut

Eine SOAP-Nachricht kann auch über mehrere Zwischenstationen zum endgültigen Ziel gelangen. Das Attribut `actor` markiert Header, welche nur für bestimmte Zwischenstationen relevant sind. Als Wert wird dabei der URI der Zwischenstation angegeben. Nach der Verarbeitung des Headers wird dieser aus der SOAP-Nachricht entfernt.

`mustUnderstand`-Attribut

Durch dass `mustUnderstand`-Attribut wird festgelegt, ob das Verarbeiten eines Headers optional (`mustUnderstand="0"`; Standard) oder obligatorisch (`must-`

Understand="1") ist. Falls bei einem Wert von „1“ die Verarbeitung fehlschlägt, so muss ein Fehler zurückgegeben werden.

[Gun02]

Body

Die eigentlichen Nutzdaten der Nachricht werden innerhalb des Body-Elements definiert. Der Body ist obligatorisch und muss direkt nach dem Header stehen. Sollte der Header nicht existieren, so ist der Body das erste Element im Envelope.

Fault-Element Neben den Nutzdaten wird der Body auch für die Fehlerdiagnose verwendet. Hierfür definiert SOAP das Kindelement Fault. Dieses darf maximal einmal in einem Body enthalten sein. Als Kindelemente bietet SOAP standardmäßig die 4 Elemente faultcode, faultstring, faultactor und detail.

faultcode-Element Das faultcode-Element gibt einen Fehlercode an, welcher für die maschinelle Behandlung des Fehlers verwendet werden kann. Die Angabe des faultCode-Elements ist bei Verwendung von Fault Pflicht. Als Fehlercodes werden standardmäßig VersionMismatch, MustUnderstand, Client und Server bereitgestellt. VersionMismatch gibt an, dass bei der Verarbeitung ein ungültiger Namensraum gefunden wurde. Wenn ein Header-Element mit mustUnderstand ="1" nicht verarbeitet werden konnte, dann wird der Fehlercode MustUnderstand verwendet. Client wird verwendet, wenn die gesendete Nachricht falsch gebildet wurde oder nicht die nötigen Informationen zur Verarbeitung besaß. Wenn der Fehler nichts mit der SOAP-Nachricht selbst, sondern mit seiner Verarbeitung zu tun hat, so wird Server verwendet.

faultstring-Element Das Element faultstring gibt eine menschenlesbare Version des Fehlers an. Die Angabe dieses Elements ist ebenfalls obligatorisch.

faultactor-Element Der Verursacher einer Fehlers wird durch die Angabe eines URI innerhalb des optionalen faultactor-Elements abgegeben.

detail-Element Das detail-Element gibt detaillierte Informationen über Fehler bei der Verarbeitung des Bodys an. Es darf nur in diesem Falle verwendet werden und ist dann sogar Pflicht. Innerhalb des detail-Elements können mehrere Kindelemente verwendet werden.

[W3C00]

2.3.5. REST

Der Begriff REST (REpresentational State Transfer) beschreibt einen Architekturstil für Hypermediasysteme, welcher sich insbesondere auf das World Wide Web bezieht. Durch REST lassen sich unter Nutzung von Web Standards (z.B. HTTP, URL, HTML) Web Services realisieren. Diese Art von Web Services wird beispielsweise von Suchmaschinen oder Online-Shops verwendet. In PegelSuite wird dieser Architekturstil für die Realisierung eines Web Services benötigt. Durch einen Link in einer E-Mail, soll der Status eines Alarms änderbar sein.

Im Folgenden wird die Nutzung als Web Service anhand eines kleinen Beispiels erläutert und anschließend die Architektur sowie deren Elemente vorgestellt.

[Fie00]

Beispiel eines RESTful Web Services

Ein Beispiel für die Verwendung von REST sind sogenannte *RESTful* (so werden die Implementationen einer REST-Architektur genannt) Web Services. Diese basieren auf der Verwendung des HTTP-Protokolls, welches sich um die Übertragung der Nachrichten kümmert und mit seinen Methoden (z.B. GET, POST, PUT und DELETE) ein einheitliches Interface zur Verfügung stellt.

Möchte man den Standard-Web-Service aus den vorherigen Abschnitten durch einen RESTful Web Service ersetzen, so könnte eine Anfrage an den Server folgendermaßen aussehen:

Eine Anfrage wird von einem Webbrowser durch Aufruf der URI `http://www.pegel-kaub.de/getWaterGaugeData/getWaterGaugeDataRequest?time=201001261753000100` (Zeitformat zur Vermeidung von Sonderzeichen verändert) an den Server gesendet. Der zugehörige HTTP-Request ist in Listing 2.12 dargestellt.

```
1 GET /getWaterGaugeData/getWaterGaugeDataRequest
2   ?time=201001261753000100 HTTP/1.1
3 Host: www.pegel-kaub.de
```

Listing 2.12: REST-Beispiel: HTTP Request

Der Server verarbeitet diese Anfrage und sendet als Repräsentation ein HTML-Dokument, innerhalb einer HTTP-Response, an den Client zurück. Diese ist in Listing 2.13 dargestellt.

```
1  HTTP/1.1 200 OK
2  Content-Type: text/xml; charset="utf-8"
3  Content-Length: 266
4
5  <?xml version="1.0" encoding="UTF-8"?>
6  <m:GetWaterGaugeDataResponse
7   xmlns:m="http://www.pegel-kaub.de/watergaugedata.wsdl">
8   <m:value>169.5</m:value>
9   <m:unit>cm</m:unit>
10 </m:GetWaterGaugeDataResponse>
```

Listing 2.13: REST-Beispiel: HTTP Response

Die Response-Nachricht enthält neben der Repräsentation (XML; Zeilen 5 - 10) zusätzlich Metadaten, wie den Medientyp und die Kodierung (Zeile 2), sowie die Länge der Nachricht (Zeile 3). Diese Antwortnachricht wird vom Browser auf der Client-Seite interpretiert und dargestellt.

Merkmale an eine REST-Architektur

Der REST-Architekturstil ist ein Hybrid mehrerer netzwerkbasierter Architekturstile und lässt sich anhand der folgenden Merkmale definieren:

- Client-Server-Architektur
- Geschichtetes System
- Zustandslosigkeit
- Cache
- Einheitliches Interface
- Code-On-Demand (Optional)

Client-Server Der *Client-Server-Architekturstil* ermöglicht das Prinzip der Trennung der Belange. Durch die Trennung von User-Interface (Client) und Datenspeicher (Server) steigt die Portabilität des User-Interfaces über mehrere Plattformen sowie die Skalierbarkeit der Server-Komponenten.

Geschichtetes System REST-Architekturen sind in Schichten aufgebaut. Dabei kann eine Schicht nur mit der Schicht genau über bzw. unter ihr kommunizieren. Dies ermöglicht bei einer Client-Server-Struktur z.B. das Hintereinanderschalten mehrerer Server oder Clients und damit eine Aufgabenverteilung über verschiedene Netzwerke oder Prozessoren. Jeder dieser Schichten ist es möglich den Inhalt einer Nachricht sinnvoll zu verändern, da diese selbstbeschreibend sind.

Zustandslosigkeit Wie auch das HTTP-Protokoll ist eine REST-Architektur *zustandslos*. Eine Anfrage an den Server muss somit alle benötigten Informationen, die zur Ausführung

nötig sind, enthalten. Die Verwendung von Cookies oder ähnlichem ist in REST nicht vorgesehen. Zustandslosigkeit vereinfacht die Server-Komponenten, indem die Verwaltung der Zustände entfällt. Durch die Menge an Informationen, die mitgeteilt werden müssen, wird das Netzwerk allerdings mehr belastet und die Performance sinkt.

Cache

Um dem entgegenzuwirken, erlaubt REST das Benutzen von *Caches*. In einer Antwort vom Server müssen Ressourcen als cachebar oder nicht cachebar gekennzeichnet werden. Sind Ressourcen cachebar, so kann bei einer erneuten Anfrage die Antwort aus dem Cache gelesen werden, anstatt den Server erneut zu kontaktieren. Ein Nachteil ist allerdings, dass der Inhalt des Caches bei einer erneuten Anfrage von der derzeitigen Antwort des Servers abweichen kann, was die Zuverlässigkeit senkt. Werden häufig die selben Anfragen an den Server geschickt, so kann dieser die Antwort ebenfalls aus einem Cache holen anstatt diese neu zu generieren oder selbst Anfragen an weitere Server zu senden (siehe „Geschichtetes System“).

Einheitliches Interface

Ein wichtiges Merkmal einer REST-konformen Architektur ist die Verwendung eines *einheitlichen Interfaces* (z.B. die HTTP-Methoden). In diesem Punkt unterscheidet sich REST auch von anderen netzwerkbasierten Architekturstilen. Ein einheitliches Interface vereinfacht die gesamte Architektur. Die Austauschbarkeit von Implementationen wird ebenfalls vereinfacht, da diese vollkommen unabhängig von der Schnittstelle sind. Ein solches Interface senkt allerdings die Effizienz, da die Informationen standardisiert übertragen werden, anstatt über ein speziell auf die Anwendung zugeschnittenes Interface.

Code-On-Demand

Das Code-On-Demand-Prinzip von REST ist optional. Es erlaubt den Clients die Erweiterung der Funktionalität durch das Ausführen von Applets oder Scripts. Dadurch werden einfachere Clients ermöglicht, da viele Funktionen nicht von vornherein implementiert werden müssen.

[Fie00]

Elemente einer REST-Architektur

Die Elemente einer REST-Architektur unterteilen sich in *Daten-Elemente*, *Komponenten* und *Konnektoren*. Bei einem Daten-Element handelt es sich um eine Information, welche von einer Komponente durch einen Konnektor übertragen bzw. empfangen wird. Eine Komponente ist eine Software-Einheit, welche zusammen mit ihrem internen Zustand die Transformation von Daten über ein Interface bereitstellt. Bei einem Konnektor handelt es sich um einen abstrakten Mechanismus, welcher die Kommunikation, Koordination und Kooperation zwischen

Komponenten regelt. Das Zusammenspiel dieser drei Elemente wird in Abbildung 2.4 nochmals dargestellt: Komponenten (z.B. Browser, Web Server) kommunizieren über Konnektoren (z.B. HTTP) durch Daten-Elemente (z.B. HTML + HTTP-Header).

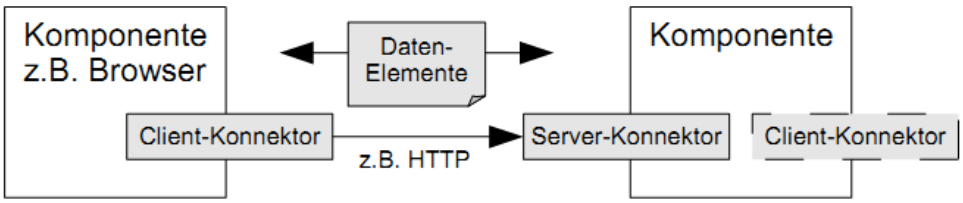


Abbildung 2.4.: Architektur-Elemente des REST-Architekturstils [Sch08]

Daten-Elemente Die Daten-Elemente beschreiben alle Elemente einer Nachricht. Man unterscheidet zwischen den 6 Daten-Elementen *Ressource*, *Ressourcen-Bezeichner*, *Repräsentation*, *Repräsentations-Metadaten*, *Ressourcen-Metadaten* und *Kontrolldaten*. Als Ressource bezeichnet man eine Menge von Entitäten. Dies können z.B. Dokumente, Bilder, ein temporärer Service etc. sein. Eine Ressource wird durch einen Ressourcen-Bezeichner (z.B durch eine URI) eindeutig identifiziert. Komponenten führen auf diesen Ressourcen Aktionen aus. Um dabei den Status zu speichern sowie für den Transfer zwischen den Komponenten werden Repräsentationen (z.B. HTML, JPEG) verwendet. Eine Repräsentation ist im Grunde eine Folge von Bytes, welche durch Repräsentations-Metadaten (z.B DTD, Medientyp) genauer beschrieben werden. Bei Repräsentationen muss es sich um Hypermedia-Daten, wie z.B. HTML handeln). Dadurch ist es möglich, z.B. durch Links zwischen Ressourcen zu navigieren. Antwortnachrichten können sowohl Repräsentations-Metadaten als auch Ressourcen-Metadaten enthalten. Letztere enthalten Informationen über die Ressource, welche nicht spezifisch für die derzeitige Repräsentation sind (z.B. Alternativen). Die Kontrolldaten beschreiben den Zweck der Nachricht zwischen den Komponenten. Diese enthalten beispielsweise Informationen wie etwa das Cache-Verhalten. Durch die Gesamtheit aller Daten-Elemente ist eine Nachricht selbstbeschreibend.

Komponenten Bei den Komponenten wird zwischen *User-Agent*, *Ursprungs-Server*, *Proxy* und *Gateway* unterschieden. Ein User-Agent initiiert eine Anfrage und ist auch der endgültige Empfänger der Antwort. Der Ursprungs-Server hat als endgültiger Empfänger direkten Zugriff auf eine Ressource und empfängt die Anfragen über einen Server-Konnektor. Durch eine Proxy-Komponente ist es möglich Interfaces anderer Services abzukapseln, Daten zu transformieren, die Performance zu steigern oder die Sicherheit zu erhöhen. Bei einem Gateway handelt es sich um eine Komponente die ein Netzwerk oder einen Ursprungs-Server verwendet um Interfaces anderer Services abzukapseln, Daten zu transformieren, die Performance

zu steigern oder die Sicherheit zu erhöhen. Der Unterschied zwischen Proxy und Gateway ist, dass ein Client bei einem Proxy die Wahl hat, ob er diesen verwendet oder nicht.

Konnektoren

Konnektoren kapseln die Implementation der Komponenten von der Übertragung der Repräsentationen ab und stellen das Interface für die Übertragung bereit. Sie lassen sich in 5 Typen unterteilen: *Client*, *Server*, *Cache*, *Resolver* und *Tunnel*. Die beiden essentiellen Konnektorentypen sind Client und Server. Sie unterscheiden sich darin, dass ein Client eine Anfrage initiiert, während der Server auf eine wartet und bei Erhalt auf diese antwortet. Es ist durchaus möglich, das eine Komponente sowohl Client- als auch Server-Konnektoren enthält (vgl. Abbildung 2.4). Ein Cache kann sowohl zu einem Client- oder Server-Interface gehören und dient dazu Informationen zwischenzuspeichern anstatt eine erneute Anfrage (Client) zu stellen oder eine Antwort erneut generieren zu müssen. Üblicherweise wird ein Cache im Adressraum des Konnektors implementiert, der diesen verwendet. Ein Resolver übersetzt einen Ressourcen-Bezeichner in eine Netzwerkadresse. Um eine Kommunikation durch Blockaden wie Firewalls schleusen zu können wird ein Tunnel verwendet.

[Fie00]

WADL

Wie auch ein typischer Web Service benötigt ein RESTful Web Service eine maschinell verarbeitbare Beschreibung (vgl. WSDL; Abschnitt 2.3.1). WADL[5] steht für Web Application Description Language und dient dazu Web-Anwendungen maschinell verarbeitbar zu beschreiben.

WADL-Beispiel

Eine mögliche WADL-Definition des im letzten Abschnitt beschriebenen Beispiels eines RESTful Web Services ist in Listing 2.14 dargestellt. In der ersten Zeile werden, wie bei XML-Dokumenten üblich, Verarbeitungsinformationen angegeben. In der zweiten Zeile beginnt die eigentliche WADL-Definition, welche durch das Wurzelelement `application` eingeleitet wird. Dieses definiert auch den Namensraum der WADL-Definition. Anschließend werden in den Zeilen 3 - 5 die Format-Definitionen angegeben. Dies geschieht durch die Referenz (`include`-Element) auf das Schema `watergaugedata.xsd` in Zeile 4. Die Zeilen 6 - 20 definieren die Ressourcen. `ressources` umschließt dabei alle Ressourcen und gibt die Basis-URI an. Die erste Ressource wird in den Zeilen 7 - 19 definiert. Der Pfad zu dieser Ressource wird durch das `path`-Attribut angeben. Als Kindelement enthält diese Ressource eine weitere Ressource (Zeilen 8 - 18). Das Kindelement `method` (Zeilen 9 - 17) gibt die verwendete HTTP-Methode (`name`-Attribut) an.

[5] Namensraum: `http://wadl.dev.java.net/2009/02`

Dieses enthält die Kindelemente `request` (Zeilen 10 - 13) und `response` (Zeilen 14 - 16). Das Kindelement `param` (Zeilen 11 und 12) von `request` definiert den Parameter `time` (`name`-Attribut). Dieser ist vom Typ `xsd:int` (`type`-Attribut). Der Parameterstil (`style`-Attribut) ist `query`. Dies ist der im Web übliche Stil durch die GET-Methode Parameter zu übergeben. Die Antwort auf die Anfrage ist eine Repräsentation als HTML-Dokument (`mediaType="text/xml"` gesetzt), welche in Zeile 15 beschrieben wird.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <application xmlns="http://wadl.dev.java.net/2009/02">
3   <grammars>
4    <include href="http://www.pegel-kaub.de/watergaugedata.xsd" />
5   <grammars>
6   <resources base="http://www.pegel-kaub.de/">
7    <resource path="getWaterGaugeData">
8     <resource path="getWaterGaugeDataRequest">
9      <method name="GET" id="getWaterGaugeDataRequest">
10      <request>
11       <param xmlns:xsd="http://www.w3.org/2001/XMLSchema"
12        type="xsd:int" style="query" name="time" />
13      </request>
14      <response>
15       <representation mediaType="text/xml"/>
16      </response>
17     </method>
18    </resource>
19   </resource>
20  </resources>
21  </application>
```

Listing 2.14: WADL-Beispiel

Elemente einer WADL-Beschreibung

Unabhängig von den gerade angebrachten Beispiel muss eine WADL-Datei die im Folgenden beschriebene Struktur einhalten.

Das Wurzelelement einer WADL-Beschreibung ist `application`. Es enthält die Kindelemente:

1. `doc` (0 oder mehr)
2. `grammars` (optional)
3. `resources` (0 oder mehr)
4. 0 oder mehr der folgenden:
   - `ressource_type`
   - `method`
   - `representation`

- param

Die Dokumentation der WADL-Datei geschieht durch das doc-Element. Es kann als Kindelement eines jeden WADL-Elements vorkommen. Es ist auch möglich, mehrere doc-Elemente im selben Kindelement anzugeben, jedoch müssen diese verschiedene Sprachen (Attribut xml:lang) besitzen.

Das Element grammars dient als Container für Format-Definitionen für den Datenaustausch. Diese lassen sich als Kindelemente definieren oder können durch das include-Element referenziert werden. Die Definitionssprache selbst ist nicht vorgeschrieben. Die Definition kann z.B. durch XML Schema geschehen.

include besitzt das Attribut href vom Typ xsd:anyURI, welches eine URI zu einer Format-Definition angibt. Es kann als Kindelement von grammars verwendet werden.

resources dient als Container für Beschreibung der Ressourcen, welche die Applikation bereitstellt. Es besitzt das Attribut base. Dieses gibt die Basis-URI, welche für jedes der resource-Kindelemente gilt, an.

Eine Menge von Ressourcen wird durch das Element resource angegeben. Diese werden jeweils durch eine URI identifiziert, welche einem gemeinsamen Muster folgt. Die möglichen Kindelemente von resource sind doc, param, method und resource. Jedes dieser Elemente darf 0 oder mehrmals vorkommen.

Das Element resource_type beschreibt die Methoden, welche das Verhalten eines Ressourcentyps definieren. Ein resource_type-Element kann dabei durchaus mehreren Ressourcen zugeordnet werden. Als Kindelemente sind 0 oder mehr doc-, param-, method- oder resource-Elemente möglich.

method beschreibt die Ein- und Ausgabe einer HTTP-Methode. Diese können dann einer Ressource zugeordnet werden. Das method-Element kann eine Definition einer Methode enthalten oder durch das href-Attribut auf eine solche verweisen. Eine Referenz kann als Kindelement von resource verwendet werden. Eine Definition hingegen kann nicht nur als Kind von resource vorkommen, sondern auch als Kindelement von application. Als Kindelemente besitzt method die Elemente doc (0 oder mehr), request und response (0 oder mehr).

Das request-Element beschreibt die Eingabe, welche der HTTP-Methode hinzugefügt werden muss. Es kann 0 oder mehr der folgenden Kindelemente enthalten: doc, representation und param.

Die möglichen Antworten einer Methode werden durch `response`-Elemente angegeben. Als Kindelemente enthält `response` 0 oder mehr `doc`-, `representation`- und `param`-Elemente.

Eine Repräsentation einer Ressource wird durch das Element `representation` beschrieben. Bei `representation` kann es sich, wie auch bei `method`, sowohl um eine Referenz als auch um eine Definition handeln. Eine Referenz ist dabei nur als Kindelement von `request` und `response` zulässig. Eine Definition hingegen kann auch als Kind von `application` vorkommen. Kindelemente von `representation` können jeweils 0 oder mehr `doc`- und `param`-Elemente sein.

Das Element `param` beschreibt eine parametrisierte Komponente des Elternelements und kann sowohl eine Referenz (durch `href`) als auch eine Definition sein. Eine Definition kann als Kindelement von `resource`, `application`, `request`, `response` und `representation` vorkommen. Kindelemente von `param` können 0 oder mehr `doc`-Elemente, 0 oder mehr `option`-Elemente oder ein optionales `link`-Element sein.

Das Element `option` definiert einen möglichen Wert für einen Parameter.

Zu guter Letzt gibt es noch das Element `link`. Dieses wird verwendet um Links zu Ressourcen in Repräsentationen zu identifizieren. `link` ist ein Kindelement von `param` und kann selbst das Kindelement `doc` beliebig häufig enthalten.

[W3C09b]

2.4. WS-BPEL

Viele der Prinzipien einer SOA (siehe Abschnitt 2.2.1) legen nahe, dass Services möglichst klein sein und nur eine elementare Aufgabe übernehmen sollten. Um größere Aufgaben zu erfüllen, müsste man jedoch mehrere Services ausführen. Für diese Aufgabe wurde die Sprache WS-BPEL (WS-Business Process Execution Language) entwickelt.

Orchestrierung

Bei BPEL handelt es sich um eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen durch Web Services. Die Zusammenstellung mehrerer Web Services zu einem Geschäftsprozess wird als *Orchestrierung* bezeichnet. Eine BPEL-Orchestrierung ist selbst wieder ein Web Service und kann in weiteren Orchestrierungen als einzelner Service verwendet werden. [Vas07]

Neben dem Aufruf verschiedener Web Services unterstützt BPEL auch die Manipulation von XML Daten, parallele Ausführung, Schleifen und bedingte Ausführung [Vas07].

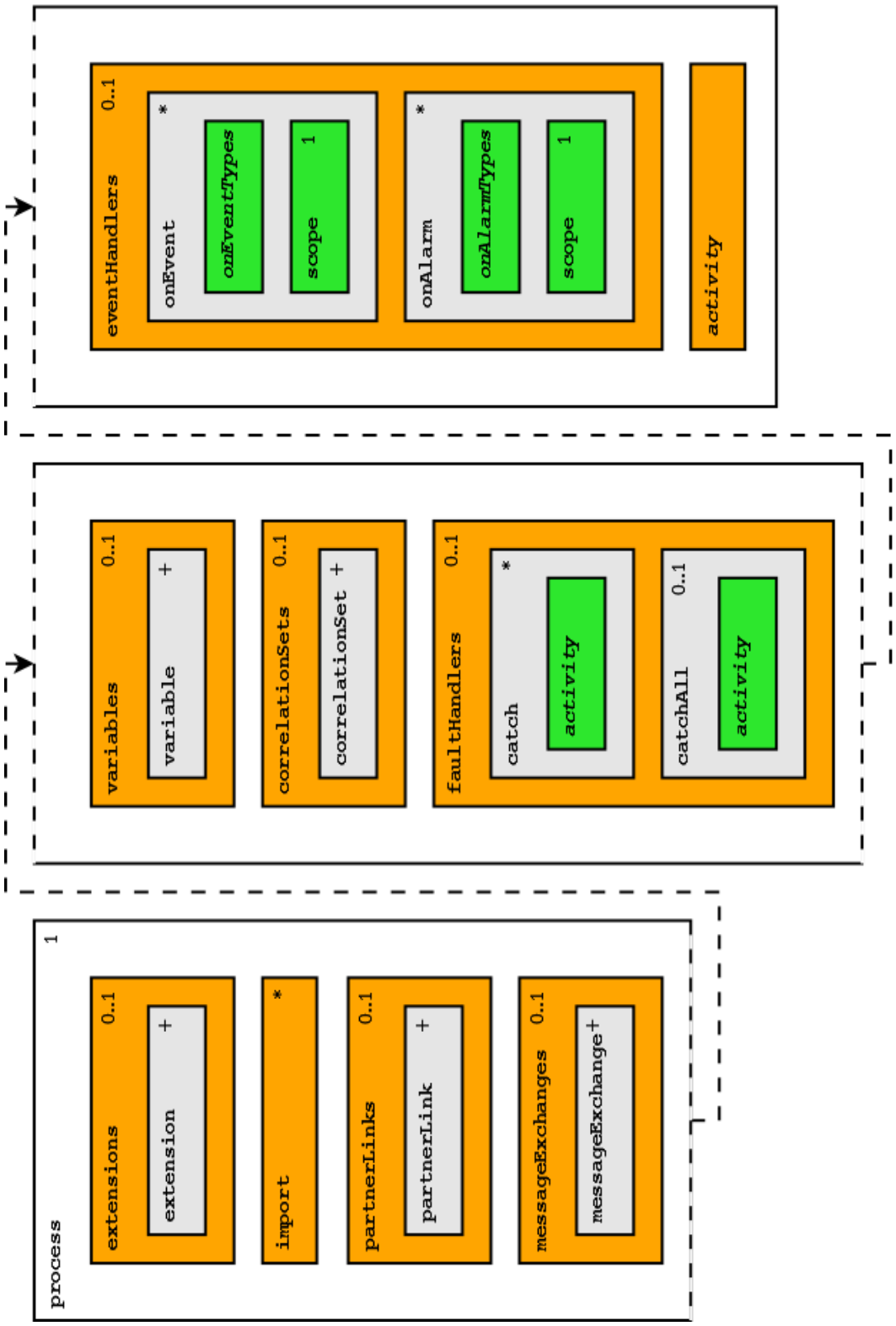


Abbildung 2.5.: Die Struktur eines BPEL-Prozesses

Die Struktur eines BPEL-Prozesses

Die eigentliche Beschreibung des Geschäftsprozesses geschieht in einer BPEL-Datei. Die Struktur einer einfachen BPEL-Datei ist in Abbildung 2.5 dargestellt. Die kursiv gedruckten Namen bezeichnen keine Elementnamen, sondern Gruppen von Elementen. Eine BPEL-Datei besteht aus dem Root-Element `process` und den Kindelementen `extensions`, `import`, `partnerLinks`, `messageExchanges`, `variables`, `correlationSets`, `faultHandlers`, `eventHandlers` sowie der Elementgruppe *`activities`*.

In Abschnitt 2.4.2 - Abschnitt 2.4.11 werden die einzelnen Elemente einer BPEL-Datei genauer erläutert. Zuvor wird das Grundprinzip von BPEL jedoch an einem Beispiel beschrieben.

BPEL-Engine

Für die Ausführung von BPEL-Dateien wird eine so genannte *BPEL-Engine*, wie etwa Apache ODE[6]), benötigt. Diese dient als Laufzeitumgebung für die BPEL-Prozesse und ist für die Interpretation und Ausführung der BPEL-Dateien zuständig [Bra09].

[OAS07]

2.4.1. Ein einführendes Beispiel

In Anhang B.1 ist ein komplettes Beispiel einer BPEL-Orchestrierung gezeigt, welches auf den folgenden Seiten abschnittsweise besprochen wird. Dies besteht aus einer WSDL-Datei (siehe Anhang B.1.1) zur Beschreibung der Schnittstelle. Der Prozess selbst wird anschließend in einer BPEL-Datei (siehe Anhang B.1.2) angegeben.

Das Beispiel zeigt den Prozess `simpleInvoke`, der den Web Service `getTimeSeries` von `http://www.xhydro-test.de` aufruft. Die grafische Notation des Prozesses, wie sie der BPEL-Designer[7] von Eclipse[8] verwendet, ist in Abbildung 2.6 dargestellt.

Der Prozess besteht aus einer Sequenz (`mainSequence`) von Befehlen (Aktivitäten). `receiveInput` wartet dabei auf eine eingehende Nachricht, welche den Prozess startet. Mittels der Aktivität `assignInvokeInput` werden die Werte der Variablen für den nachfolgenden Web Service Aufruf von `getTimeSeries` gesetzt. Durch `invokeGetTimeSeries` wird dieser aufgerufen. Die Antwortnachricht des Aufrufs wird mittels der Aktivität `assignReplyOutput` in die Antwortnachricht des Prozesses kopiert, welche mit `replyOutput` versendet wird.

[6] `http://ode.apache.org/` (abgerufen: 16.07.2011)
[7] `http://www.eclipse.org/bpel` (abgerufen: 16.07.2011)
[8] `http://www.eclipse.org` (abgerufen: 16.07.2011)

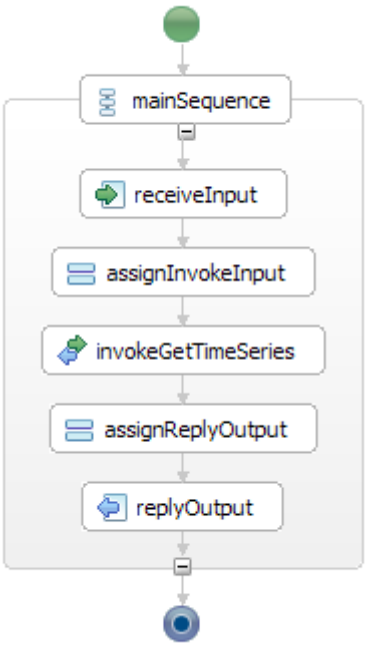


Abbildung 2.6.: Der Prozess `simpleInvoke.bpel`

`simpleInvoke.wsdl`

Wie jeder Web Service besitzt auch ein BPEL-Prozess eine WSDL-Datei (siehe Anhang B.1.1). Der Aufbau einer WSDL-Datei wurde bereits in Abschnitt 2.3.1 erläutert. Allerdings kann diese nun einige Erweiterungselemente von BPEL, wie etwa `partnerLinkType`-Elemente[9], enthalten.

Neben der eigenen WSDL-Beschreibung müssen auch alle WSDL-Dateien der aufgerufenen Services importiert werden. Dies ist notwendig um die dort definierten Nachrichten dem BPEL-Prozess zugänglich zu machen. Listing 2.15 importiert die benötigte `getTimeSeries.wsdl` für den Aufruf `invokeGetTimeSeries` (siehe Abbildung 2.6).

```
13 <import location="getTimeSeries.wsdl"
14  namespace="http://xhydro.org/tns" />
```

Listing 2.15: WS-BPEL-Beispiel (WSDL): `import`

Die WSDL-Datei des Prozesses `simpleInvoke` verwendet zudem das Element `partnerLinkType` (siehe Listing 2.16). PartnerLinks (siehe Abschnitt 2.4.5) legen die Rollen der einzelnen Kommunikationspartner fest. Diese Rollen werden im Kindelement `role` von `partnerLinkType` definiert. Sie referenzieren

[9] Namensraum: `http://docs.oasis-open.org/wsbpel/2.0/plnktype`

einen `portType`. In Listing 2.16 werden Rollen für die Kommunikation zwischen dem Aufrufer des Prozesses und dem Prozess `simpleInvoke` (Zeilen 48, 49), sowie zwischen `simpleInvoke` und dem Web Service `getTimeSeries` (Zeilen 52, 53) definiert. Die Rollen referenzieren jeweils einen PortType. Für die Kommunikationsbeziehung zwischen dem Aufrufer und `simpleInvoke` ist dies der PortType `simpleInvokePT` aus `simpleInvoke.wsdl`. Für die Kommunikationsbeziehung zwischen dem Prozess `simpleInvoke` und dem Web Service `getTimeSeries` wird der PortType `getTimeSeries_WebService` aus der Datei `getTimeSeries.wsdl` verwendet.

```
47 <plnk:partnerLinkType name="simpleInvokePLT">
48  <plnk:role name="simpleInvokeProvider"
49   portType="tns:simpleInvokePT" />
50 </plnk:partnerLinkType>
51 <plnk:partnerLinkType name="getTimeSeriesPLT">
52  <plnk:role name="getTimeSeriesProvider"
53   portType="wsdl:getTimeSeries_WebService" />
54 </plnk:partnerLinkType>
```

Listing 2.16: WS-BPEL-Beispiel (WSDL): `partnerLinkType`

simpleInvoke.bpel

Der eigentliche Prozess wird in der Datei `simpleInvoke.bpel` beschrieben. Dieser wurde anhand von Abbildung 2.6 bereits grob vorgestellt. Diese Darstellung enthielt jedoch nur den Kontrollfluss des Prozesses. Zu einer BPEL-Orchestrierung gehören jedoch auch weitere Angaben. Beispielsweise müssen die verwendeten Variablen und PartnerLinks deklariert werden.

```
1 <bpel:process name="simpleInvoke" suppressJoinFailure="yes"
2  targetNamespace=
3   "http://www.pegelsuite.de/prototypes/simpleInvoke"
4  xmlns:tns="http://www.pegelsuite.de/prototypes/simpleInvoke"
5  xmlns:bpel=
6   "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
7  xmlns:xhydrotns="http://xhydro.org/tns"
8  xmlns:xhydroreq="http://xhydro.org/ws/request">
```

Listing 2.17: WS-BPEL-Beispiel: `process`

Listing 2.17 zeigt das Wurzelelement `process`. Innerhalb dessen werden die verwendeten Namensräume angegeben. Das Attribut `suppressJoinFailure` in Zeile 1 gibt an, ob Join-Fehler unterdrückt werden sollen oder nicht (siehe Abschnitt 2.4.2).

```
10 <bpel:import location="getTimeSeries.wsdl"
11  namespace="http://xhydro.org/tns"
12  importType="http://schemas.xmlsoap.org/wsdl/" />
13 <bpel:import location="simpleInvoke.wsdl"
14  namespace="http://www.pegelsuite.de/prototypes/simpleInvoke"
15  importType="http://schemas.xmlsoap.org/wsdl/" />
```

Listing 2.18: WS-BPEL-Beispiel: `import`

Auch in die BPEL-Datei müssen die verwendeten WSDL-Dateien durch das Element `import` importiert werden. Zeilen 10 - 12 importieren die Datei `getTimeSeries.wsdl` und die Zeilen 13 - 15 die Datei `simpleInvoke.wsdl`. Das Attribut `importType` gibt den Typ des Imports (hier: WSDL) an. Nähere Informationen zu `import` werden in Abschnitt 2.4.4 gegeben.

```
17 <bpel:partnerLinks>
18  <bpel:partnerLink name="simpleInvokePL"
19   partnerLinkType="tns:simpleInvokePLT"
20   myRole="simpleInvokeProvider" />
21  <bpel:partnerLink name="getTimeSeriesPL"
22   partnerLinkType="tns:getTimeSeriesPLT"
23   partnerRole="getTimeSeriesProvider" />
24 </bpel:partnerLinks>
```

Listing 2.19: WS-BPEL-Beispiel: `partnerLinks`

Die PartnerLinks (siehe Abschnitt 2.4.5) werden unter dem Element `partnerLinks` zusammengefasst (siehe Listing 2.19). Dieses enthält die beiden PartnerLinks `simpleInvokePL` (Zeilen 18 - 20) und `getTimeSeriesPL` (Zeilen 21 - 23). Mit dem Attribut `partnerLinkType` (Zeile 19 und Zeile 22) wird der jeweilige PartnerLinkType aus der WSDL-Datei `simpleInvoke.wsdl` angegeben. Durch das Attribut `myRole` (Zeile 20) wird die Rolle des Prozesses für den PartnerLink angegeben. Das in Zeile 23 angegebene Attribut `partnerRole` gibt hingegen die Rolle des Partners an. Es können für jeden PartnerLink beide Attribute oder nur eines von beiden angegeben werden.

```
26 <bpel:variables>
27  <bpel:variable name="simpleInvokeRequest"
28   messageType="tns:simpleInvokeRequestMessage" />
29  <bpel:variable name="simpleInvokeResponse"
30   messageType="tns:simpleInvokeResponseMessage" />
31  <bpel:variable name="getTimeSeriesRequest"
32   messageType="xhydrotns:getTimeSeriesRequestMessage" />
33  <bpel:variable name="getTimeSeriesResponse"
34   messageType="xhydrotns:getTimeSeries_ResponseMessage" />
```

```
35 </bpel:variables>
```

Listing 2.20: WS-BPEL-Beispiel: `variables`

In Listing 2.20 werden die Variablen definiert (siehe Abschnitt 2.4.7). Dies geschieht durch das Element `variables`. Die Kindelemente (`variable`) definieren die Variablen `simpleInvokeRequest` (Zeilen 27, 28), `simpleInvokeResponse` (Zeilen 29, 30), `getTimeSeriesRequest` (Zeilen 31, 32) und `getTimeSeriesResponse` (Zeilen 33, 34). Durch das Attribut `messageType` wird der Variable ein Nachrichtentyp zugewiesen.

```
37 <bpel:sequence name="mainSequence">
```

Listing 2.21: WS-BPEL-Beispiel: `mainSequence`

Nach diesen Deklarationen wird der eigentliche Prozess definiert, wie er bereits in Abbildung 2.6 dargestellt ist. Dieser besteht aus verschiedenen Aktivitäten, welche in Abschnitt 2.4.11 genauer erläutert werden.

Durch die Aktivität `sequence` wird eine Sequenz eingeleitet. Die Sequenz `mainSequence`, welche in Listing 2.21 deklariert wird, enthält alle weiteren Aktivitäten des Prozesses.

```
38 <bpel:receive name="receiveInput"
39  partnerLink="simpleInvokePL" portType="tns:simpleInvokePT"
40  operation="simpleInvokeOperation"
41  variable="simpleInvokeRequest" createInstance="yes" />
```

Listing 2.22: WS-BPEL-Beispiel: `receiveInput`

Das nächste verwendete Element ist `receive`. Durch die `receive`-Aktivität wird auf das Eintreffen einer Nachricht gewartet (siehe Listing 2.22). In Zeile 39 wird ein PartnerLink (`partnerLink`) und ein PortType (`portType`) angegeben. Da der PartnerLink über den PartnerLinkType ebenfalls auf den PortType verweist, muss dieser mit dem im Element angegebenen PortType übereinstimmen. Die zugehörige Operation der WSDL-Datei wird durch das Attribut `operation` in Zeile 40 referenziert. Die in der Nachricht gelieferten Daten werden in der Variablen `simpleInvokeRequest` gespeichert (Zeile 41). Das Attribut `createInstance` gibt an, dass eine neue Instanz des Geschäftsprozesses gestartet wird.

```
42 <bpel:assign validate="no" name="assignInvokeInput">
43  <bpel:copy>
44   <bpel:from>
45    <bpel:literal xml:space="preserve">
46     <tns:getTimeSeriesRequest
```

```
47      xmlns:tns="http://xhydro.org/ws/request"
48      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
49      <tns:locationNumber>25100107</tns:locationNumber>
50      <tns:locationName>Mainz</tns:locationName>
51      <tns:parameterCode>W</tns:parameterCode>
52      <tns:from>2010-03-31T14:00:00+02:00</tns:from>
53      <tns:to>2010-03-31T14:15:00+02:00</tns:to>
54     </tns:getTimeSeriesRequest>
55    </bpel:literal>
56   </bpel:from>
57   <bpel:to variable="getTimeSeriesRequest"
58    part="getTimeSeriesRequestPart" />
59  </bpel:copy>
60 </bpel:assign>
```

Listing 2.23: WS-BPEL-Beispiel: `assignInvokeInput`

Das Element `assign` erlaubt es, Variablen einen Wert zuzuweisen. Die Aktivität `assignInvokeInput` in Listing 2.23 kopiert die Aufrufparameter für den Aufruf von `getTimeSeries` in die Variable `getTimeSeriesRequest`. Das Attribut `validate` in Zeile 42 gibt an, dass keine Änderungen an den Variablen validiert werden. Im `copy`-Element (Zeilen 43 - 59) wird der Variable `getTimeSeriesRequest` ein Wert zugewiesen. Dazu werden die Zeilen 46 - 54 innerhalb des `from`-Elements (Zeilen 44 - 56) nach (`to`; Zeilen 57, 58) `getTimeSeriesRequest` (Zeile 57), genauer gesagt in den Part `getTimeSeriesRequestPart` (Zeile 58), kopiert.

```
61 <bpel:invoke name="invokeGetTimeSeries"
62  partnerLink="getTimeSeriesPL"
63  portType="xhydrotns:getTimeSeries_WebService"
64  operation="getTimeSeries"
65  inputVariable="getTimeSeriesRequest"
66  outputVariable="getTimeSeriesResponse" />
```

Listing 2.24: WS-BPEL-Beispiel: `invokeGetTimeSeries`

Nachdem die Werte der Nachricht zugewiesen wurden, wird der Web Service `getTimeSeries` durch die `invoke`-Aktivität `invokeGetTimeSeries` aufgerufen (siehe Listing 2.24). Dieses Element gibt durch die Attribute `partnerLink`, `portType` und `operation` (Zeilen 62 - 64) die nötigen Informationen für den Aufruf des Services. Weiterhin werden die Input-Variable (Zeile 65) und die Output-Variable (Zeile 66) angegeben.

```
67 <bpel:assign validate="no" name="assignReplyOutput">
68  <bpel:copy>
69   <bpel:from part="getTimeSeries_ResponsePart"
```

```
70      variable="getTimeSeriesResponse" />
71     <bpel:to part="simpleInvokeResponseTsel"
72      variable="simpleInvokeResponse" />
73    </bpel:copy>
74  </bpel:assign>
```

Listing 2.25: WS-BPEL-Beispiel: `assignReplyOutput`

In Listing 2.25 werden die Werte der Output-Variable durch die `assign`-Aktivität `assignReplyOutput` in die Variable für die `replyOutput`-Aktivität kopiert. Dazu reicht es, die Variablen und deren Parts anzugeben, welche kopiert werden sollen (Zeilen 69 - 72).

```
75  <bpel:reply name="replyOutput" partnerLink="simpleInvokePL"
76   portType="tns:simpleInvokePT"
77   operation="simpleInvokeOperation"
78   variable="simpleInvokeResponse" />
```

Listing 2.26: WS-BPEL-Beispiel: `replyOutput`

Zu guter Letzt wird die Antwort wieder zurück an den Aufrufer gesendet (siehe Listing 2.26). Hierfür werden die Attribute `partnerLink` (Zeile 75), `portType` (Zeile 76), `operation` (Zeile 77) und `variable` (Zeile 78) benötigt.

2.4.2. Das Wurzelelement `process`

`process` ist das Wurzelelement einer BPEL-Orchestrierung. Es definiert die Attribute `queryLanguage`, `expressionLanguage`, `suppressJoinFailure` und `exitOnStandardFault`.

`queryLanguage` beschreibt die Anfragesprache, welche für den Prozess verwendet wird. Standardmäßig ist dies `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`, welches die Verwendung von XPath 1.0 innerhalb von WS-BPEL 2.0 definiert.

Durch das Attribut `expressionLanguage` wird die verwendete Ausdruckssprache des Prozesses angegeben. Dies ist standardmäßig wieder XPath 1.0 innerhalb von WS-BPEL 2.0.

Das `suppressJoinFailure`-Attribut macht es möglich Join-Fehler zu unterdrücken. Der Standardwert ist `no`. Ein Join-Fehler tritt auf, wenn das Zusammenführen von Aktivitäten (siehe Abschnitt 2.4.11) - wie etwa bei `flow` - fehlschlägt.

`exitOnStandardFault` gibt an, ob ein Prozess direkt (wie bei dem Element `exit`) beendet wird, wenn ein Fehler (außer Join-Fehler) auftritt oder ob Fehler durch einen FaultHandler aufgelöst werden können. Der Standardwert ist `no`.

[OAS07]

2.4.3. Extensions

WS-BPEL bietet durch das Element `extensions` die Möglichkeit an, die Sprache zu erweitern. Erweiterungen können beispielsweise neue Attribute oder Elemente, erweiterte Aktivitäten, Erweiterungen oder Einschränkungen zur Laufzeit sein.

Das `extensions`-Element stellt dabei eine Liste von Extensions (`extension`) dar. Durch die Angabe eines Namensraumes (`namespace`-Attribut) innerhalb des `extension`-Elements ist es möglich, die Erweiterungen zu definieren. Nach der Definition können die Elemente und Attribute des Namensraumes verwendet werden. Die Elemente in BPEL sind dabei so entworfen, dass sie durch Elemente und Attribute des neuen Namensraumes erweitert werden können [Ebe08].

Neben dem Namensraum lässt sich auch noch durch das Attribut `mustUnderstand` festlegen, ob die Erweiterung von der BPEL-Engine verarbeitet werden muss oder nicht. Ist der Wert `yes`, dann muss die BPEL-Engine diese Erweiterung unterstützen. Ist dies nicht der Fall, dann wird der gesamte Prozess abgelehnt. Bei dem Wert `no` wird die Erweiterung von der BPEL-Engine einfach ignoriert.

[OAS07]

2.4.4. Import

Eine BPEL-Orchestrierung benötigt neben ihrer eigenen WSDL-Datei auch Zugriff auf die WSDL-Dateien der aufzurufenden Web Services. Um diese sowie die Schema-Beschreibungen zu importieren, gibt es das Element `import`. Es besitzt die 3 Attribute `importType`, `namespace` und `location`.

Bei dem Attribut `importType` handelt es sich um ein Pflicht-Attribut von `import`. Es gibt den Typ des Imports (meist WSDL oder XML-Schema) an. Dies geschieht über einen URI (WSDL: `http://schemas.xmlsoap.org/wsdl`; XML-Schema: `http://www.w3.org/2001/XMLSchema`).

Das optionale Attribut `namespace` gibt den Namensraum der importierten Quelle an. Dieser muss, sofern er angegeben wurde, mit dem Standard-Namensraum der Quelle übereinstimmen.

Durch `location` wird der Ort der zu importierenden Datei angegeben. Das Attribut ist ebenfalls optional.

Trotz der Tatsache, dass lediglich `importType` ein Pflicht-Attribut ist, sollten möglichst immer alle 3 Attribute angeben werden, da sonst kein Namensraum festgelegt wird und auch nicht erwähnt wird, wo der Import zu finden ist.

[OAS07]

2.4.5. PartnerLinks

Die Kommunikation zwischen mehreren Geschäftspartnern durch Web Services wird meistens durch eine Peer-to-Peer (P2P) Verbindung realisiert. Daher ist eine Zwei-Wege-Kommunikation notwendig, so dass ein Partner sowohl in die Rolle des Clients (Requester) als auch des Servers (Provider) schlüpfen kann.

Durch *PartnerLinks* (`partnerLink`) wird die Kommunikation der Partner untereinander für beide Richtungen (falls nötig) beschrieben. Dies geschieht durch die Angabe eines *PartnerLinkTypes* (`partnerLinkType`) und der zugehörigen *Rolle* (`role`). Die genauen Beziehungen zwischen den PartnerLinks, den PartnerLinkTypes und den PortTypes werden in Abbildung 2.7 dargestellt.

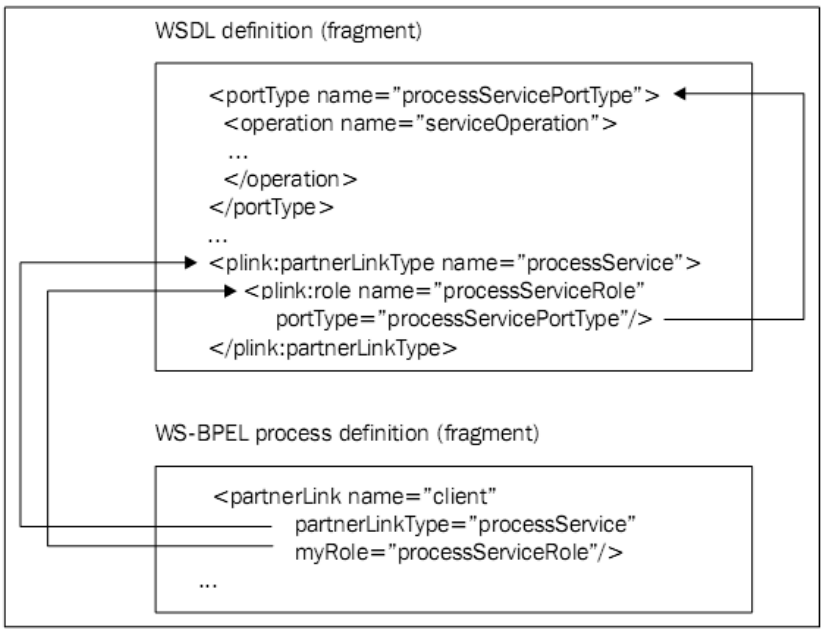


Abbildung 2.7.: Beziehungen der PartnerLinks [Vas07]

partnerLink-Element

Der Kommunikationspartner eines jeden Service-Aufrufs (Invoke) wird in WS-BPEL durch das Element `partnerLink` (Kindelement von `partnerLinks`) beschrieben. Es besitzt die folgenden wichtigen Attribute: `partnerLinkType`, `myRole`, `partnerRole` und `initializePartnerRole`.

Durch das Attribut `partnerLinkType` wird der zugehörige PartnerLinkType der WSDL-Datei angegeben.

`myRole` verweist auf die eigene Rolle in einer Kommunikationsbeziehung, während `partnerRole` einen Verweis auf die Rolle des Partners enthält. `myRole` und `partnerRole` können sowohl beide verwendet werden, als auch nur eines von beiden. Es muss jedoch mindestens eines dieser Attribute angegeben werden.

Wenn das Attribut `initializePartnerRole` auf `yes` gesetzt wird, so wird die Verbindung der Partner bereits vor der ersten Nutzung initialisiert. Bei `no` wird die Verbindung erst bei der Nutzung hergestellt.

Die verfügbaren Rollen werden in der WSDL-Datei innerhalb des `partnerLinkType`-Elements als Kindelement `role` definiert. Jede Rolle enthält einen Verweis auf den verwendeten PortType. Damit besitzt man alle notwendigen Informationen, um den Aufruf durchzuführen.

[OAS07]

2.4.6. MessageExchanges

Wenn ein Prozess mehrere eingehende, Aktivitäten aufrufende Nachrichten (z.B `receive`) und `reply`-Aktivitäten (siehe Abschnitt 2.4.11) zum selben PartnerLink und der selben Operation ausführen kann, ist es nötig, diese unterscheidbar zu machen. Dies geschieht durch das Attribut-Tupel `partnerLink`, `operation` und `messageExchange` der jeweiligen Aktivität.

Das Element `messageExchanges` (Kindelement von `definitions`) definiert eine Liste von `messageExchange`-Elementen, welche durch das Attribut `name` unterschieden werden.

Kann eine `reply`-Aktivität nicht mit einer vorangegangenen Nachricht in Verbindung gebracht werden, so muss der Fehler `bpel:missingRequest` ausgelöst werden.

[OAS07]

2.4.7. Variablen

Um Daten zwischen den einzelnen Aufrufen von Web Services zu speichern, verfügt BPEL über Variablen. Diese werden innerhalb des Elements `variables` jeweils durch ein Element `variable` definiert. Veränderungen an den Variablen können durch die Aktivität `assign` (siehe Abschnitt 2.4.11) vorgenommen werden.

Die Deklaration der Variablen kann dabei durch WSDL Nachrichten Typen (Attribut `messageType`), XML Schema Typen (Attribut `type`) oder XML Schema Elemente (Attribut `element`) geschehen. Pro Variable muss genau eine dieser Arten verwendet werden.

[OAS07]

2.4.8. CorrelationsSets

Nachrichten werden in WS-BPEL nicht an die Schnittstelle der Prozesse gesendet, sondern an Instanzen dieser Prozesse. Um feststellen zu können, welche Instanz der Empfänger der Nachricht ist, muss man der Nachricht Zusatzinformationen hinzufügen, welche die Instanz eindeutig identifizieren.

Dies wird in BPEL durch *CorrelationsSets* (`correlationSets`-Element) erreicht. Es handelt sich dabei um einen deklarativen Mechanismus für die Zuordnung der beteiligten Operationen zu einer gemeinsamen Instanz. Dieses Element darf als Kindelement von `process` und `scope` (siehe Abschnitt 2.4.11) vorkommen. `correlationSets` enthält eine Liste von `correlationSet`-Elementen. Diese enthalten eine Liste von *Nachrichteneigenschaften*, welche durch das Attribut `properties` (Leerzeichen zwischen den einzelnen Eigenschaften) angegeben werden.

Die Nachrichteneigenschaften[10] werden vorher global in der WSDL-Datei definiert. Dies geschieht durch das Erweiterungselement `property`. Für dieses Element kann ein Name (`name`-Attribut) und ein Typ (`type`-Attribut) angegeben werden. Als Typ muss ein einfacher Datentyp aus XML Schema verwendet werden. Damit diese Eigenschaft verwendet werden kann, muss ihr allerdings noch ein Eigenschaftswert zugewiesen werden. Dabei handelt es sich um Teile einer Nachricht. Die Zuweisung geschieht durch das Element `propertyAlias`. Dessen Attribute `propertyName`, `messageType` und `part` geben die nötigen Informationen an. `propertyName` gibt dabei den Namen der Nachrichteneigenschaft an, `messageType` referenziert eine Nachricht aus der WSDL-Datei und `part` einen der Nachrichtenparts.

[10] Namensraum: `http://schemas.xmlsoap.org/ws/2003/03/business-process/`

Die so definierten Korrelationen können nun innerhalb jeder Aktivität einer BPEL-Datei, welche im Stande ist Nachrichten zu senden oder zu empfangen (`receive`, `reply`, `onMessage`, `onEvent` und `invoke`), angegeben werden. Innerhalb der Aktivitäten ist es möglich, ein Kindelement `correlations` zu definieren, welches eine Liste von `correlation`-Elementen enthält. Jedes dieser Elemente gibt eine Korrelation an. Dies geschieht durch die Attribute `set`, `initiate` und `pattern` (nur bei `invoke`).

Das Pflichtattribut `set` referenziert dabei den Namen eines correlationSet-Elements.

`initiate` gibt an, ob das CorrelationSet noch instanziiert werden muss. Dabei gelten zwei Beschränkungen: *Consistency Constraint* und *Initiation Constraint*. Die *Consistency Constraint* besagt, dass nach der Instanziierung eines CorrelationSets dessen Werte nicht mehr verändert werden dürfen, solange man sich in dem Scope (Gültigkeitsbereich) befindet, für das die CorrelationSet gilt. Die *Initiation Constraint* besagt folgendes:

- Wenn der Wert von `initialize` auf `yes` gesetzt wird, dann muss die zugehörige Aktivität versuchen, das CorrelationSet zu instanziieren.
  - Wenn das CorrelationSet bereits instanziiert ist, muss der Standardfehler `bpel:correlationViolation` geworfen werden.
- Wenn der Wert von `initialize` auf `join` gesetzt wird, dann muss die zugehörige Aktivität das CorrelationSet instanziieren, wenn dies nicht schon geschehen ist.
  - Wenn dass CorrelationSet bereits instanziiert ist und die Consistency Constraint verletzt wurde, muss der Standardfehler `bpel:correlationViolation` geworfen werden.
- Wenn der Wert von `initialize` auf `no` (Standard) gesetzt wird, dann darf die zugehörige Aktivität das CorrelationSet nicht instanziieren.
  - Wenn das CorrelationSet vorher nicht schon instanziiert wurde, muss der Standardfehler `bpel:correlationViolation` geworfen werden.
  - Wenn das CorrelationSet bereits instanziiert wurde, aber die Consistency Constraint verletzt wurde, muss der Standardfehler `bpel:correlationViolation` geworfen werden.

Für die Aktivität `invoke` existiert zusätzlich noch das Attribut `pattern`. Dieses gibt an, ob die Zuordnung für die eingehende Nachricht (`response`), die ausgehende (`request`) oder für beide (`request-response`) gilt. Die Verwendung

von `pattern` ist bei einem Zwei-Wege-Aufruf Pflicht und bei einem Ein-Weg-Aufruf verboten.

[OAS07]

2.4.9. FCT-Handlers

Unter dem Begriff der FCT-Handler werden *FaultHandler*, *CompensationHandler* und *TerminationHandlers* zusammengefasst. Diese sind eng miteinander verbunden und dienen der Rollback von Scopes bei Fehlern oder vorzeitiger Terminierung. Die Fehlerbehandlung dient in WS-BPEL dazu unfertige und fehlgeschlagene Aktivitäten, des Scopes in dem der Fehler aufgetreten ist, umzukehren.

FaultHandler

Das Element `faultHandlers` kann als Kindelement eines Scopes (mit `process` als globaler Scope) vorkommen. `faultHandlers` kann die Elemente `catch` und `catchAll` enthalten. Diese beschreiben, wie ein aufgetretener Fehler behandelt wird. Dies geschieht innerhalb der Elemente durch Aktivitäten.

Jedes `catch`-Element beschreibt genau einen Fehlertyp. Es enthält die Attribute `faultName`, `faultVariable`, `faultMessageType` und `faultElement`. Das Attribut `faultName` definiert dabei den Namen des Fehlers. Für die Angabe fehlerspezifischer Informationen kann das Attribut `faultVariable` verwendet werden. Es gibt eine Variable an, in welcher die Informationen gespeichert werden. Diese Variable muss extra für dem FaultHandler deklariert werden und ist auch nur innerhalb von diesem sichtbar. Wird eine Fehlervariable angegeben, so muss dessen Typ ebenfalls, entweder als WSDL Nachrichtentyp (`faultMessageType`) oder XML Schema Typ (`faultElement`), angegeben werden.

Alle Fehler, welche nicht durch das `catch`-Element behandelt werden, werden durch `catchAll` abgefangen. Wurde dieses nicht angegeben, so wird der Standard-FaultHandler aufgerufen.

[OAS07]

CompensationHandler

Wenn in einem Scope ein Fehler auftritt, kann es sein, dass bereits einige untergeordnete Scopes komplett ausgeführt wurden. Für die Fehlerbehandlung kann es notwendig sein, die Aktionen dieser Scopes rückgängig zu machen. Dies geschieht durch die *CompensationHandler*.

Während die Fehlerbehandlung des Scopes, in dem der Fehler aufgetreten ist, durch den FaultHandler ausgeführt wird, geschieht dies für die bereits abgeschlossenen, untergeordneten Scopes durch die CompensationHandler `compensateScope` und `compensate`. Diese Elemente dürfen ledilich als Kindelemente von `catch`, `catchAll`, `compensationHandler` und `terminationHandler` vorkommen.

`compensate` macht dabei alle Aktionen der inneren Scopes rückgängig, während bei `compensateScope` nur die Aktionen eines speziellen Scopes rückgängig gemacht werden. Dies geschieht durch die Angabe des Attributs `target`.

[OAS07]

TerminationHandler

Sollte ein Scope vorzeitig terminiert werden (z.B. durch `exit`), dann lässt sich durch einen *TerminationHandler* auf dessen Terminierung Einfluss nehmen. Wurde dieser nicht definiert, so wird der Standard-TerminationHandler aufgerufen. Ein TerminationHandler kann nur aufgerufen werden, wenn der Scope sich im normalen Modus (keine Fehlerbehandlung etc.) befindet. Während der Ausführung des TerminationHandlers werden alle Fault- und EventHandler deaktiviert. Sollte es dennoch zu einem Fehler kommen, so wird dieser nicht weitergegeben. Dies liegt daran, dass der übergeordnete Scope bereits einen Fehler behandelt oder sich ebenfalls in einem Terminationsprozess befindet oder eine `forEach`-Schleife durch die `completionCondition` bereits vorzeitig beendet wurde.

[OAS07]

2.4.10. EventHandlers

Neben den FCT-Handlern gibt es in BPEL noch zwei Arten von *EventHandlern*: `onEvent` und `onAlarm`. Diese werden als Kindelemente von dem Element `eventHandlers` definiert, wobei mindestens eines dieser Elemente vorhanden sein muss. Die Aktionen beim Auslösen eines EventHandlers werden innerhalb eines eigenen Scopes (Kindelement `scope` von `onEvent` oder `onAlarm`) definiert. Es können dabei auch mehrere EventHandler nebenläufig ausgeführt werden.

Der onEvent-Handler wird durch eingehende Nachrichten ausgelöst. Dessen Pflicht-Attribute sind partnerLink und operation. Weitere optionale Attribute sind portType, messageExchange, variable, messageType und element. Das partnerLink-Attribut enthält den Namen des zugehörigen PartnerLinks und operation den Namen der Operation. Das Attribut partnerLink ist optional, muss - sofern es verwendet wird - allerdings den Namen des PortTypes enthalten, welcher durch das myRole-Attribut des angegeben PartnerLinks referenziert wird. Das Attribut messageExchange referenziert einen MessageExchange (siehe Abschnitt 2.4.6). Die Variable, in welcher die eingehende Nachricht gespeichert werden soll, wird durch das Attribut variable referenziert. Der Type der Nachricht kann entweder durch die Referenz auf einen Nachrichtentyp (messageType) oder durch eine neue Elementtyp-Definition (element) angegeben werden.

Eine Alternative zur Angabe einer Variable ist das Element fromParts. Dieses enthält eine Liste von fromPart-Elementen, welche es erlauben die einzelnen Parts einer Nachricht in verschiedene Variablen zu kopieren. Dafür ist die Angabe des Parts (part-Attribut) und der Variable (toVariable-Attribut) nötig.

Weiterhin lassen sich für den EventHandler onEvent durch das Kindelement correlations noch CorrelationSets (siehe Abschnitt 2.4.8) angeben.

Zum Auslösen eines Events nach Ablauf einer bestimmten Zeit wird der onAlarm-Handler verwendet. Dieser enthält die Kindelemente for, until, repeatEvery und scope. Durch for wird das Event ausgelöst, sobald die angegebene Zeit verstrichen ist. Diese Startet, sobald das zugehörige Scope betreten wird. until hingegen löst einen Event zu einem angegeben Zeitpunkt aus. repeatEvery löst einen Event periodisch aus, bis der zugehörige Scope (sowie alle eingeschlossenen Scopes) verlassen wird. Wird repeatEvery in Verbindung mit for oder until verwendet, wird die periodische Auslösung verzögert, bis diese ausgelöst wurden. Erst dann startet der Timer für repeatEvery. Durch das optionale Attribut expressionLanguage von for, unit und repeatEvery kann die Ausdruckssprache angegeben werden, in welcher die Zeitangaben gemacht wurden.

2.4.11. Aktivitäten

Der eigentliche Geschäftsprozess wird schließlich durch die verschiedenen Aktivitäten beschrieben. Diese werden im Folgenden aufgelistet und genauer erläutert.

- receive
- reply

- invoke
- assign
- throw
- exit
- wait
- empty
- sequence
- if
- while
- repeatUntil
- forEach
- pick
- flow
- scope
- compensate
- compensateScope
- rethrow
- validate
- extensionActivity

Durch receive wird auf das Eintreffen einer Nachricht gewartet. Üblicherweise wird receive zusammen mit dem Attribut createInstance="yes" verwendet, um den Geschäftsprozess zu starten. Neben diesem Attribut existieren noch variable um die verwendete Variable festzulegen, messageExchange um eine Reply- mit einer Receive-Aktivität zu assoziieren, sowie operation um die aufzurufende Operation zu bestimmen.

Die reply-Aktivität sendet eine Antwortnachricht auf eine vorangegangene receive-Aktivität.

Um Partner Web Services aufzurufen existiert das invoke- Konstrukt. Anhand des partnerLink-Attributes lässt sich der aufzurufende Service bestimmen. Weitere wichtige Attribute sind inputVariable und outputVariable, welche die Variablen für Sende- und Empfangsnachricht angeben. Die aufzurufende Operation wird durch das Attribut operation angegeben.

Das assign-Element wird verwendet um Variablen Werte zuzuweisen. Dies kann durch das einfache kopieren eines Wertes oder durch XPath 1.0 Ausdrücke geschehen.

Fehler werden durch throw geworfen. Durch das Attribut faultVariable wird eine Variable für Informationen über den Fehler angegeben.

Ein Geschäftprozess, in welchem sich das `exit`-Element befindet, wird sofort beendet.

Um eine bestimmte Zeit (Kindelement `for`) oder bis zu einem bestimmten Zeitpunkt (Kindelement `until`) zu warten wird das `wait`-Element verwendet.

Das `empty`-Element bewirkt nichts. Man kann es aber zum Beispiel zur Synchronisation von nebenläufigen Aktivitäten nutzen.

Eine Sequenz von Aktivitäten wird durch `sequence` erzeugt. Die Elemente innerhalb dieses Elements werden nacheinander, in der Reihenfolge des Auftretens, abgearbeitet.

Für Fallunterscheidungen steht das `if`-Konstrukt zur Verfügung. Die Bedingung, welche für die Ausführung erfüllt sein muss wird durch das Kindelement `condition` angegeben. Wenn die Bedingung nicht erfüllt ist, so werden die Aktivitäten unter `else` ausgeführt. Um weitere Bedingungen zu überprüfen gibt es noch das Konstrukt `elseif`.

Solange die Bedingung des Kindelements `condition` wahr ist, werden die Aktivitäten innerhalb des `while`-Elements wiederholt.

Um eine Schleife solange zu wiederholen, bis eine Bedingung (Kindelement `condition`) wahr wird, verwendet man `repeatUntil`. Die Bedingung wird am Ende eines Schleifendurchlaufs getestet, weshalb die Schleife mindestens einmal ausgeführt wird.

`forEach` wird verwendet um Aktivitäten wiederholt auszuführen. Die Kindelemente `startCounterValue` und `finalCounterValue` geben dabei den Startwert bzw. den Endwert der Ausführungen an. Ist das Attribut `parallel="yes"` von `forEach` gesetzt, so werden alle N Ausführungen gleichzeitig durchgeführt. Durch das Kindelement `completionCondition` kann eine `forEach`-Anweisung vorzeitig beendet werden, sollte die Bedingung wahr werden. Diese steht im Kindelement `branches` und enthält einen Integer-Wert, welcher die Verarbeitung abbricht, sobald diese Anzahl Durchläufe abgearbeitet wurde. Sein Attribut `successfulBranchesOnly` gibt dabei an, ob nur erfolgreich abgearbeitete Durchläufe gezählt werden.

Durch das `pick`-Element wird mit der Ausführung der inneren Aktivitäten solange gewartet, bis eine Nachricht eintrifft (Kindelement `onMessage`) oder ein Timeout (Kindelement `onAlarm`) erfolgt.

Innerhalb des `flow`-Elements werden die Aktivitäten nebenläufig ausgeführt. Das Kindelement `links` kann dabei Abhängigkeiten zwischen den Aktivitäten definieren.

`scope` schafft eine Umgebung mit eigenen `partnerLinks`, `messageExchanges`, `variables`, `correlationSets`, `faultHandlers`, `compensationHandlers`, `terminationHandlers` und `eventHandlers`.

Um bei einem Fehler Änderungen in allen inneren Scopes rückgängig zu machen, verwendet man `compensate`. Dieses Element kann nur innerhalb von `catch` oder `catchall`-Elementen eines `faultHandlers`, `compensationHandlers` und `terminationHandlers` auftreten.

Im Gegensatz zu `compensate` wird durch `compensateScope` nur ein innerer Scope rückgängig gemacht. Dieses Element darf ebenfalls nur innerhalb der Elemente `catch` oder `catchall`-Elementen eines `faultHandlers`, `compensationHandlers` und `terminationHandlers` auftreten.

Um einen abgefangenen Fehler weiter zureichen wird `rethrow` verwendet. Es darf nur innerhalb von `catch` oder `catchall`-Elementen eines `faultHandlers` stehen.

Um Variablen gegen ihre Datentypdefinition zu validieren, wird `validate` verwendet.

`extensionActivity` wird verwendet um neue Aktivitätstypen in WS-BPEL einzuführen.

[OAS07]

2.5. XHydro

Bei XHydro handelt es sich um eine XML-Sprache mit dem Ziel einen Standard für den organisationsübergreifenden und wirtschaftlichen Austausch von Zeitreihen inklusive der beschreibenden Metainformationen zu schaffen. Sie wurde von den Firmen *Kisters* und *disy* im Auftrag der Bundesanstalt für Gewässerkunde entwickelt.

[Bun08]

2.5.1. Zeitreihen

Als Zeitreihe bezeichnet man im Allgemeinen eine über die Zeit aufgetragene Reihe von Werten [Tre10]. Eine solche Zeitreihe nennt man *isochron*, falls die Zeitabstände äquidistant sind. Im Falle von XHydro handelt es sich um Zeitreihen zur Übertragung von Pegeldaten.

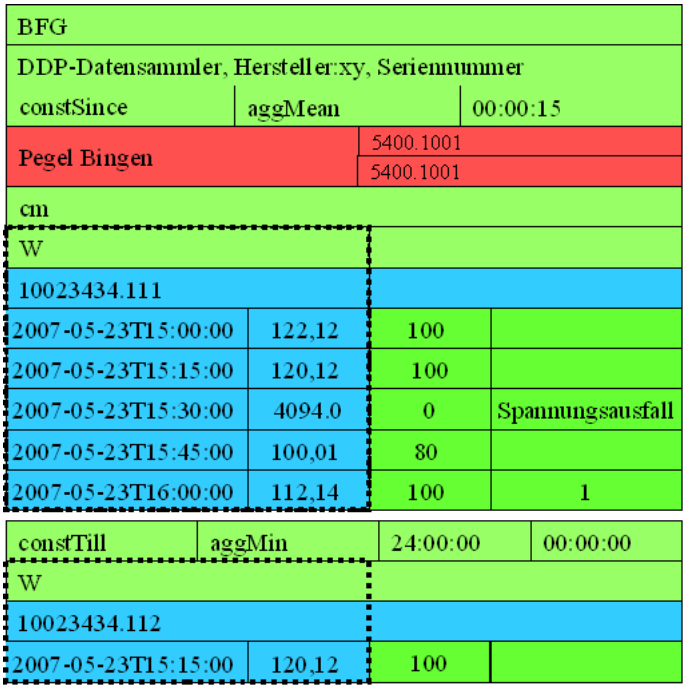


Abbildung 2.8.: Elemente einer Zeitreihe [Kd07c]

Abbildung 2.8 zeigt das Zeitreihenmodell von XHydro. Es lässt sich grob in *Zeitreihenkern* (blau) und *Metadaten* (grün und rot) unterteilen. Die einzelnen Elemente der Zeitreihe werden in Abschnitt 2.5.3 genauer erläutert.

2.5.2. Designentscheidungen

Bei der Entwicklung von XHydro spielte das Design eine wichtige Rolle. Die Anforderungen an das Format waren sehr spezifisch und sollten auf die bestmögliche Weise realisiert werden.

Die wichtigsten Anforderungen sind:

- Verschlüsselbarkeit
- Signierbarkeit
- Komprimierbarkeit
- Selbstbeschreibbarkeit
- Interoperabilität
- Erweiterbarkeit
- Anpassbarkeit
- Flexibilität
- Wiederverwendbarkeit
- Validierbarkeit

[Bun09]

In den folgenden Unterkapiteln werden die Designentscheidungen erläutert und Anhand der Anforderungen begründet.

Signatur, Verschlüsselung und Komprimierung

Für die Signatur, Verschlüsselung und Komprimierung liegen bereits Standards vor, welche diese spezifizieren. Durch die Verwendung dieser Technologien wird das XHydro Schema vereinfacht.

Signatur

Die Signatur kann durch den Standard *XML-Signature* (siehe [W3C08b]) realisiert werden. Dieser sieht mehrere Varianten der Signatur vor: *Detached*, *Enveloped* und *Enveloping*

Bei der Variante `Detached` wird die Signatur unabhängig vom ursprünglichen Inhalt gespeichert. Dies kann in einer separaten Datei (remote) oder im gleichen XML-Dokument (local) erfolgen. *Enveloped* bedeutet, dass die Signatur innerhalb des zu signierenden Elements liegt und *Enveloping*, dass der ursprüngliche Inhalt von der Signatur umfasst wird.

Aufgrund der Tatsache, dass XHydro-Dokumente selbstbeschreibend sein sollen, ist es nicht möglich die Variante *Detached (remote)* zu verwenden. Empfohlen wird die Verwendung von *Enveloping* [Kd07b]. Dadurch wird der Inhalt des Dokuments durch eine Signatur nicht verändert und kann auch ganz ohne Signatur verschickt werden.

Verschlüsselung

Die Verschlüsselung eines Dokumentes erfolgt durch XML-Encryption (siehe [W3C02]) und kann in verschiedenen Granularitätsebenen erfolgen:

- Willkürliche Daten
- Ganzes XML-Dokument
- Elemente
- Inhalt eines Elements: Kindelemente
- Inhalt eines Element: Character Data

Der Nachteil aller Verschlüsselungsformen ist es, dass das Dokument erst entschlüsselt werden muss, bevor man es weiterverarbeiten kann.

Komprimierung

Zur Komprimierung von XHydro wurden verschiedene Maßnahmen getroffen. Die eigentlichen Komprimierung kann mit einer gängige Komprimierungsmethode (z.B. ZIP) durchgeführt werden. Empfohlen wird die Verwendung der HTTP-Komprimierung, da diese transparent für die Endgeräte ist [Kd07b].

Sollte die Komprimierung des Dokuments nicht ausreichen, so kann zur Übertragung binäres XML verwendet werden. Hierbei werden die Daten in ein spezielles binäres Format (z.B. ASN.1) umgewandelt und übertragen. Obwohl Werkzeuge zur Transformation von XML nach ASN.1 existieren, sind diese nicht für alle Plattformen verfügbar und meistens mit Lizenzkosten verbunden. Vom Gebrauch von binärem XML ist daher abzuraten [Kd07b].

Ebenso wurden weitere Maßnahmen zur Verkleinerung der Dokumentgröße getroffen:

- Keine unnötige Wiederholung von Elementen
- Kurze Elementnamen
- Unnötige Elemente vermeiden
- Namensraum-Präfixe vermeiden

Einige Elemente wiederholen sich häufig innerhalb einer Zeitreihe. Bei isochronen Zeitstempeln ist es zum Beispiel sinnvoller einen Start-Zeitstempel und den Zeitabstand anzugeben, anstatt für jeden Wert einen Zeitstempel. Durch kurze Elementnamen, wird zwar die Lesbarkeit erschwert, jedoch die Performance erheblich verbessert. Um unnötige Elemente zu vermeiden ist es notwendig, dass die meisten Elemente des Schemas optional sind.

Wenn in einem XML-Dokument verschiedene Namensräume verwendet werden, so müssen üblicherweise auch die Namensraum-Präfixe verwendet werden. Dies lässt sich dadurch vermeiden, dass die `elementFormDefault`- und `attributeFormDefault`-Optionen global auf `unqualified` definiert werden. Dies bewirkt, dass der Namensraum eines Elements an seine Nachfahren weitervererbt wird. Definiert man nun viele lokale Elemente und Attribute, so wird die Angabe von Namensraumpräfixen minimiert. Nachteil dieses Ansatzes ist es, dass lokale Elemente später nicht mehr verändert werden können. Dies widerspricht der Anforderung, dass das Schema erweiterbar sein soll. Die wichtigsten Elemente sollten daher global definiert werden. Ein Präfix kann dennoch vermieden werden, indem das Schema lediglich einen Namensraum definiert, welcher als Standard-Namensraum definiert wird.

Diese Maßnahmen sollen die Größe des Dokuments bei unkomprimierter Übertragung reduzieren. Bei eingeschalteter Komprimierung sind diese Maßnahmen jedoch unwesentlich.

[Kd07b]

Interoperabilität von Web Services

Um die Interoperabilität von Web Services zu gewährleisten ist es wichtig, dass XHydro sich an den Standard der Web Services Interoperability Organization[11] hält (siehe [Web06]). Kapitel 4.8 dieses Standards schreibt folgendes vor:

- Eine Schema-Beschreibung kann jedes Konstrukt des XML Schema 1.0 verwenden.
- Eine Schema-Beschreibung muss XML Schema 1.0 als Basis für benutzerdefinierte Datentypen und Strukturen verwenden.

XHydro sollte daher ausschließlich auf den standardisierten Konstrukten von XML Schema 1.0 beruhen. Weiterhin ist es erlaubt, neue Konstrukte auf Basis der standardisierten Konstrukte einzuführen. Eine inhaltliche Einschränkung von XHydro besteht daher nicht.

[Kd07b]

Erweiterbarkeit, Modularität und Modellierungstechnik

Modularisierung

Um die Anpassbarkeit, Wiederverwendbarkeit und Erweiterbarkeit zu erhöhen, ist es nötig, die verschiedenen fachlichen Aspekte in separate Schema-Dateien (im Folgenden *Module* genannt) zu gliedern. Diese lassen sich dann in der Haupt-Schema-Datei kombinieren.

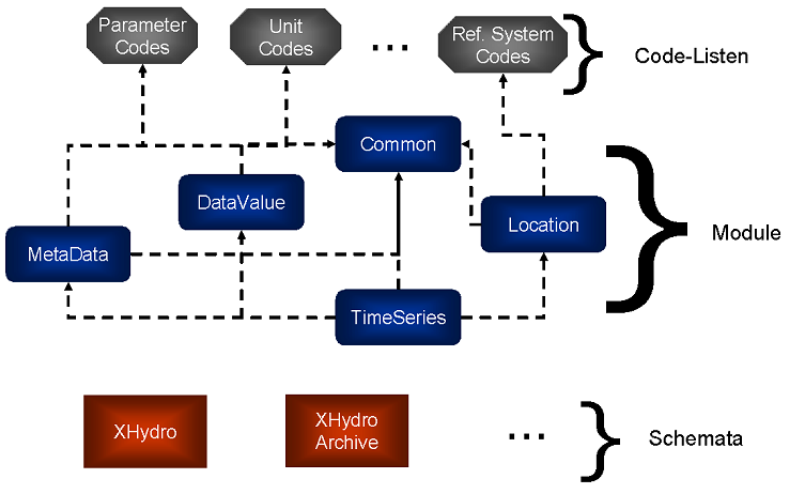


Abbildung 2.9.: Übersicht über die XHydro-Module [Kd07b]

Um die Anforderung zu erfüllen, dass XHydro lediglich einen Namensraum pro Schema verwendet, sollten die Schema-Module keine eigenen Namensräume definieren. So ist es ebenfalls möglich, diese Module in verschiedenen Schemata

[11] `http://www.ws-i.org` (abgerufen: 16.07.2011)

wiederzuverwenden. Dabei ist allerdings zu beachten, dass Abhängigkeiten zwischen den Modulen bestehen (siehe Pfeile in Abbildung 2.9). Daher müssen auch die weiteren benötigten Module in dem Schema eingebunden werden. Diese sind allerdings durch andere austauschbar. Ein Beispiel hierfür ist in Abbildung 2.10 gezeigt:

TimeSeries.xsd verwendet den Typ `dataValueQualityType` (fett markiert), welcher in *MetaData.xsd* definiert wird. Zusammengeführt werden alle benötigten Module in der Datei *XHydro.xsd*. Möchte man den Typ `dataValueQualityType` anders definieren, so tauscht man einfach die Schema-Datei *MetaData.xsd* durch eine andere, welche den selben Typ definiert, aus (hier: *MetaDataStrict*) (rot markiert). Diese wird dann statt der alten Datei in der Hauptdatei eingefügt (hier: *XHydroArchive*). Der Typ `dataValueQualityType` bleibt dabei gleich. Es müssen lediglich die rot markierten Stellen geändert werden.

Schema-Organisation

Um die Anforderungen an die Erweiterbarkeit zu erfüllen, ist neben der Modularisierung auch die richtige Schema-Organisation wichtig. Zu beachten ist hierbei, dass lokale Elemente, Attribute und Inhaltsmodelle nur innerhalb des Elternelements gültig sind und daher in anderen Modulen nicht referenziert werden können.

In XHydro werden alle Elemente, mit Ausnahme der Wurzelelemente, lokal und alle Typen global definiert. Hierdurch ist es möglich, die Verwendung von Namensräumen im Schema zu minimieren, da die Namensräume für lokale Elemente weggelassen werden können. Allerdings es es nicht mehr möglich, die Elemente direkt zu referenzieren und dadurch in anderen Schemas in anderer Kombination zu verwenden.

Wiederverwendbarkeit

Die Wiederverwendbarkeit der Elemente des Schemas war ebenfalls eine wichtige Anforderung bei der Entwicklung von XHydro.

Üblicherweise lassen sich die Elemente auf drei Arten wieder verwenden:

1. Durch Komposition von vorhandenen Elementen und Gruppen lassen sich neue Elemente oder Gruppen erstellen. Dies ist in XHydro nicht möglich, da die meisten Elemente lokal definiert sind.
2. Die global definierten Typen und Element-Gruppen lasse sich zu neuen Typdefinitionen oder Gruppen-Definitionen erweitern bzw. einzuschränken. Dies ist aufgrund der globalen Deklaration aller Typen in XHydro möglich.
3. Wiederverwendung, durch Benutzung des `redefine`-Konstrukts, ermöglicht es, Typ- und Elementdefinitionen zu ersetzten. Dies gilt jedoch nur für Schemas, welche den gleichen Namensraum definieren. In XHydro ist diese Voraussetzung erfüllt.

Abbildung 2.10.: Austausch einer Schema-Datei [Kd07b]

Joker-Elemente

Um nicht für jede neue, bisher nicht vorhergesehene Information, gleich das Schema ändern zu müssen, wurden in XHydro spezielle Joker-Elemente eingeführt. Am Ende jedes komplexen Elements lässt sich ein `any`-Schema-Konstrukt verwenden. Weiterhin gibt es für jedes einfache oder komplexe Element das `anyAttribute`-Schema-Konstrukt.

Beide Konstrukte benötigen die Parameter `namespace="##other"` und `processContent="lax"`. Der Parameter `namespace="##other"` wird zur Vermeidung nichtdeterministischer Content-Modelle benötigt. Hersteller werden dadurch gezwungen, ihre Erweiterungen in einem anderen Namensraum zu definieren. Der Parameter `processContent="lax"` macht die Schema-Definitionen für die Erweiterungen optional. Dadurch ist es möglich, den offiziellen Teil des XML-Dokuments auch einlesen und validieren zu können, auch wenn die herstellerspezifischen Schemas nicht verfügbar sind.

Die Joker-Elemente ermöglichen es, fast überall im Dokument herstellerspezifische Informationen einzufügen. Einzige Voraussetzung hierfür ist die Verwendung eines anderen Namensraumes.

Elemente mit variablem Inhalt

Des Weiteren wurde der Inhalt vieler Elemente des XHydro Schemas ewusst nicht genau spezifiziert. Der Grund hierfür ist die Vielzahl an Möglichkeiten einen Sachverhalt darzustellen. Eine Ortsangabe für einen Messort kann beispielsweise ein Punkt, eine Linie, eine Fläche oder ein Subraum eines dreidimensionalen Raumes sein.

XHydro verwendet das `<choice>`-Konstrukt zur Konstruktion von Elementen mit variablem Inhalt. Ein Vorteil des `<choice>`-Konstrukts ist die Unabhängigkeit der variablen Elemente voneinander. Die Elemente benötigen keinerlei gemeinsame Typabstammung. Nachteilig ist bei diesem Ansatz jedoch, dass sich die Liste der Elemente später nur durch eine Änderung des Schemas erweitern lässt.

Code-Listen

XHydro bietet die Möglichkeit, Listen von Codes (z.B. `dataValueQualityRemark` (Kurzform: `vqr`) oder `timeStampQualityRemark` (Kurzform: `tsqr`)) in das Schema zu integrieren. Diese lassen sich als `enumeration` definieren und mit jedem XML Schema fähigen Parser validieren, ob die Codes der jeweiligen offiziellen Code-Liste entnommen wurden. Um die Einführung eigener Codes zu gewährleisten, wurde neben dem Element für die offiziellen Code-Listen jeweils ein weiteres Element eingeführt, dessen Inhalt frei spezifiziert werden kann. Es bietet weiterhin optionale Attribute, die es erlauben, die Code-Liste, aus der der Code stammt, genauer zu beschreiben. Die Namensgebung der beiden Elemente unterscheidet sich lediglich durch ein vorangestelltes `x` (`xvqr` statt `vqr`) bei den

offiziellen Codelisten. Die Einführung eines zusätzlichen Elements für die Einbindung eigener Code-Listen gewährleistet auch weiterhin eine Validierung des Dokuments.

[Kd07b]

Versionierung

Bei der Versionierung des Schemas wird zwischen großen und kleinen Versionssprüngen (Haupt- und Nebenversion) unterschieden. Eine Version wird in der Form `x.y` gekennzeichnet, wobei `x` die Hauptversion und `y` die Nebenversion bezeichnet.

Als Nebenversion gelten Änderungen, die keine validen Dokumente invalidieren, also abwärtskompatibel sind. Als kleine Versionssprünge gelten daher das Hinzufügen eines neuen optionalen Elements oder Attributs. Alle anderen Änderungen würden valide Dokumente invalidieren und gelten daher als große Versionssprünge.

Neben der Angabe der Versionsnummer durch das Attribut `version` werden große Versionsänderungen im Namensraum deutlich. Dieser enthält eine monatsgenaue Datumsangabe, wann der Versionssprung erfolgt ist. Dieser Namensraum repräsentiert eine Webseite, von welcher das Schema heruntergeladen werden kann und wo eine Beschreibung des Schemas zu finden ist. Für eine bestimmte Hauptversion sollte dort immer die neueste Nebenversion zu finden sein. Dies gewährleistet eine Verwendung der neuesten Nebenversion bei neuen Dokumenten. Ältere Dokumente werden dadurch nicht beeinflusst, da diese sich auch durch die neueste Nebenversion validieren lassen.

[Kd07b]

2.5.3. Der XML Schema Entwurf von XHydro

Abbildung 2.11 zeigt den XML Schema Entwurf von XHydro. In diesem werden die Anforderungen an die Datenübertragung im Pegelwesen (siehe Abschnitt 2.5.2) umgesetzt. Im Kern lässt sich das Modell für verschiedene Typen von Zeitreihen wiederverwenden, jedoch sind die im Modell spezifizierten Metadaten auf hydrologische Zeitreihen zugeschnitten.

In Abbildung 2.11 ist der wiederverwendbare *Zeitreihenkern* in blau und gelb und die spezifischen *Metadaten* in grün und rot dargestellt. Zu beachten ist, dass in Abbildung 2.11 nicht alle Code-Listen des fertigen Modells enthalten sind. Diese wurden erst im XML Schema (siehe Abschnitt 2.1.3) hinzugefügt.

[Kd07a]

Der Zeitreihenkern

Zeitreihenkern

Der Zeitreihenkern (siehe Abbildung 2.12) besteht aus den Elementen `timeSeriesList`, `timeSeries`, `exchangeIds`, `exchangeId`, `timedDataElement`, `timeStamp`, `isochron` und `dataValue`.

Das Root-Element ist `timeSeriesList`. Es enthält 1 bis n Zeitreihen (`timeSeries`), welche jeweils eine Liste von 1 bis n Elementen (timedDataElement) enthalten. Jedes `timedDataElement` besteht aus einem Zeitstempel (`timeStamp`) und 1 bis n Werten `dataValue`. Im Falle einer isochronen Zeitreihe (`isochron`) kann anstatt des Zeitstempels auch ein zeitlicher Abstand (`distance` angegeben werden. In diesem Falle wird lediglich ein Start-Zeitstempel (`startTimeStamp`) benötigt.

Das Element `exchangeIds` enthält eine Liste mit 1 bis n Austauschnummern (`exchangeId`). Austauschnummern ermöglichen ein einfacheres Zuordnen von importierten Zeitreihen zu internen Zeitreihen. Um Austauschnummern verschiedener Organisationen zu unterscheiden, werden die Austauschnummern als Kombination eines Schlüssels (`exKey`) mit einem Wert (`exValue`) angegeben.

Datentypen

Die Werte einer Zeitreihe können von verschiedenem Typ sein (Abbildung 2.13). Daher enthält `value` die Subklassen `dFloatValue`, `dIntegerValue`, `dStringValue`, `dBinaryValue`, `dBoolValue` und die allgemeine Subklasse `dAnyValue`.

[Kd07a]

Metadaten

Die Metadaten teilen sich in 3 Kategorien auf, welche die unterschiedlichen Aspekte einer Zeitreihe und ihrer Daten beschrieben:

- Beschreibung der Herkunft der Daten
- Art der Daten
- Qualität der Daten

Sie enthalten Elemente, welche die verschiedenen Code-Listen repräsentieren und bestehen aus einer vorgegebenen Menge von Werten, die jederzeit erweitert werden kann.

Beschreibung der Herkunft der Daten

Die Herkunft der Daten (siehe Abbildung 2.14) kann im Modell über eine Geräteangabe (`device`) und eine Ortsangabe (`location`) beschrieben werden. Eine

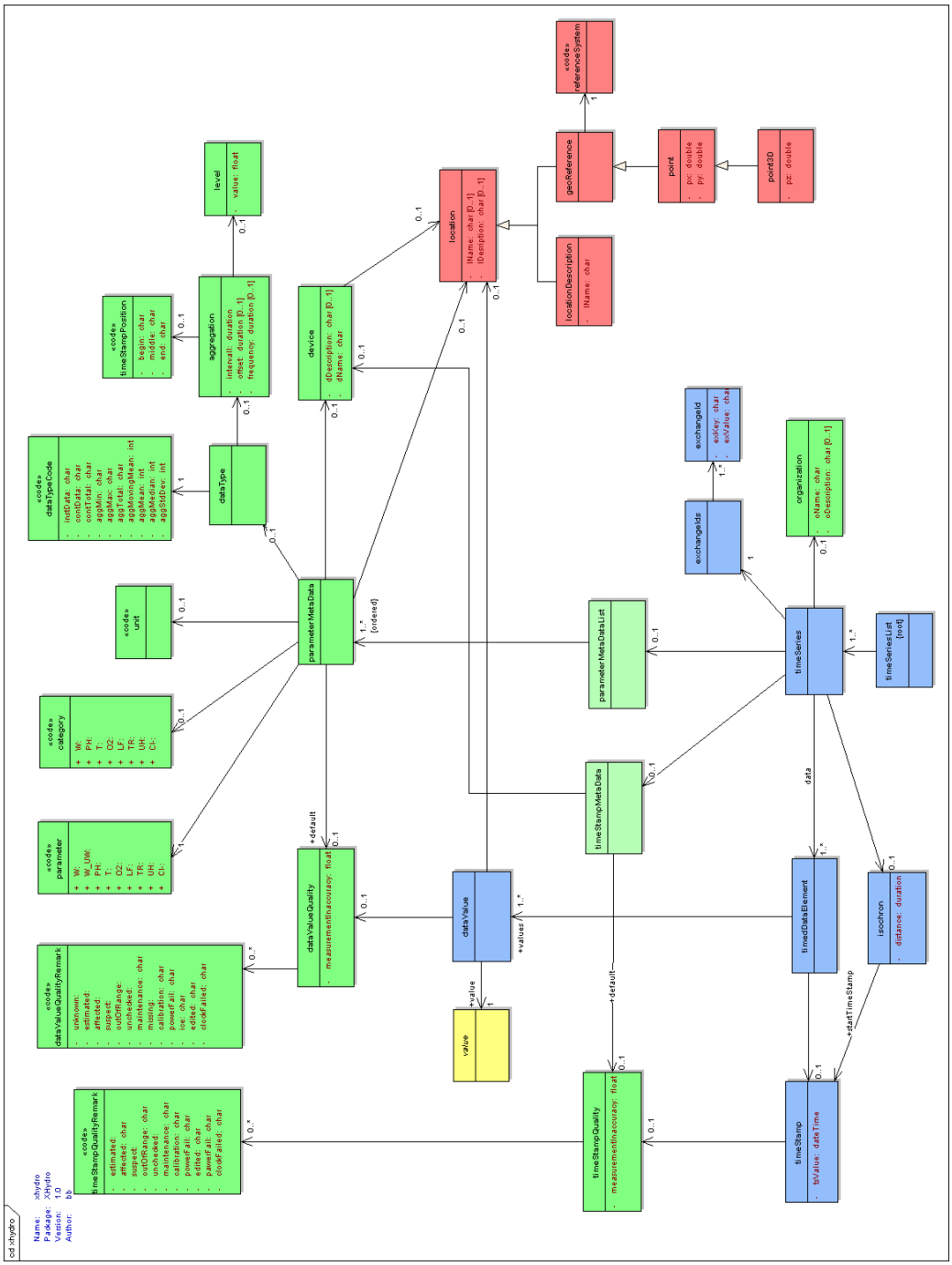


Abbildung 2.11.: Der XHydro Schema Entwurf [Bun07b]

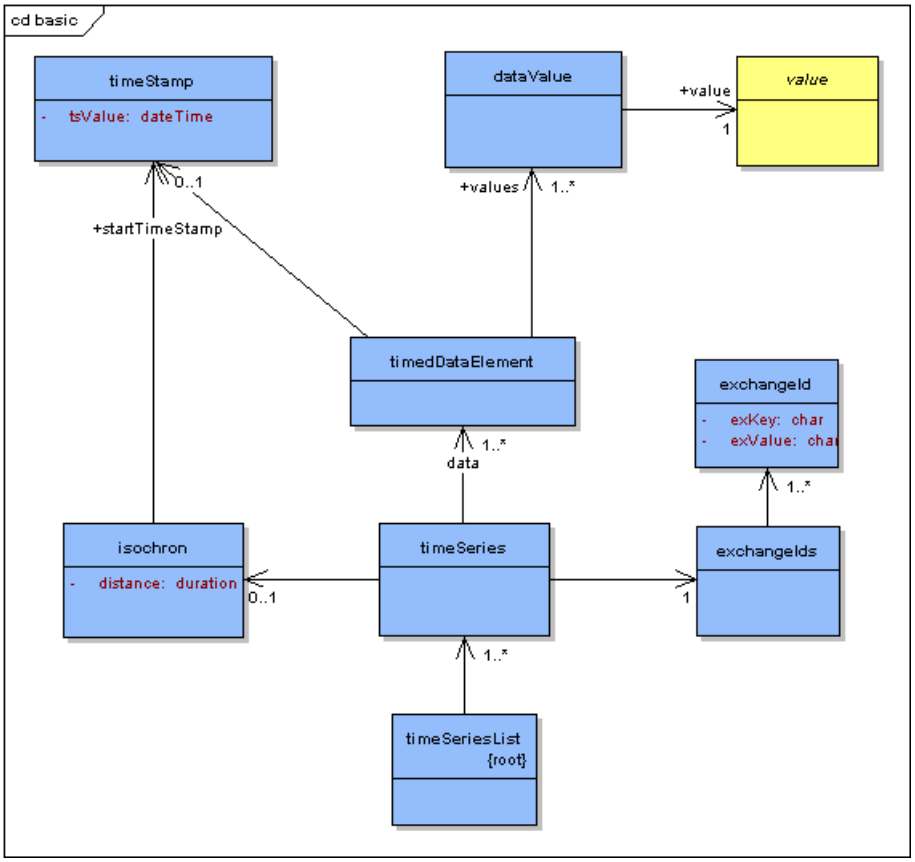


Abbildung 2.12.: Der Zeitreihenkern von XHydro [Bun07b]

Ortsangabe kann durch die Basisklasse `location`, unter Angabe eines Namens (`lName`) und einer Ortsbeschreibung (`lDescription`) erfolgen. Die Ortsangabe kann ebenfalls durch eine der beiden Unterklassen `locationDescription` oder `geoReference` erfolgen. `locationDescription` erlaubt lediglich die Angabe eines Namens, während sich mit `geoReference` eine Georeferenz in Form eines zwei- (`point`) oder dreidimensionalen (`point3D`) Punktes, einer Linie (`line`), einer Fläche (`area`), oder eines Raumes (`space`) angeben lässt. Die Beschreibung geschieht anhand eines Wertes aus der Code-Liste `referenceSystem`.

`organisation` gibt die zu den Zeitreihen gehörige Organisation mit Namen (`oName`) und Beschreibung (`oDescription`) an.

Art der Daten Die Art der Daten wird durch eine Messgröße in Kombination mit einer Bildungsregel beschrieben.

Messgröße Die Messgröße wird durch die drei Codelisten `parameter`, `category` und `unit` in Abbildung 2.15 beschrieben. `parameter` beschreibt die Art der Daten. `category` hingegen beschreibt die Kategorie der Daten. Im Grunde ist dies lediglich eine allgemeinere Form von `parameter`. Die Einheit der gemessenen Daten

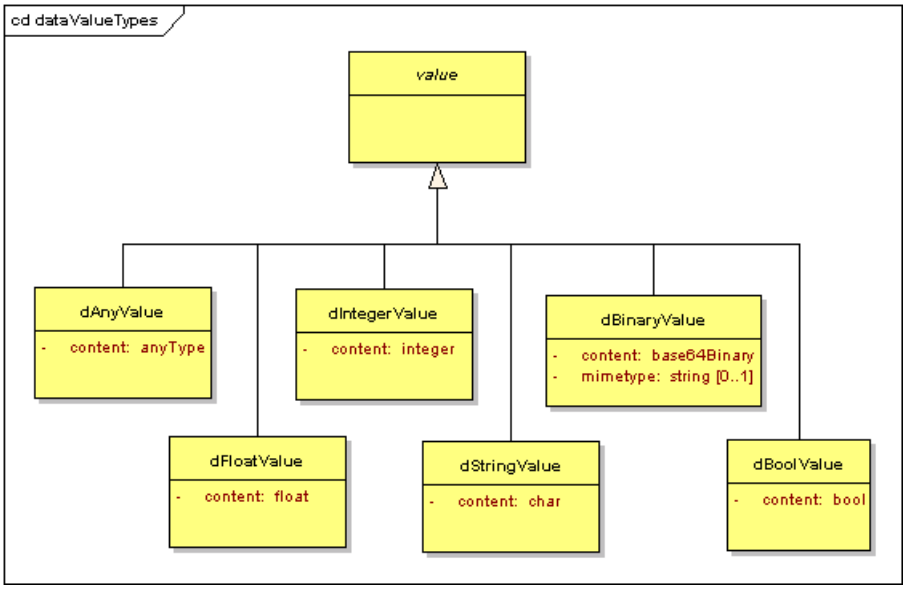


Abbildung 2.13.: Die Datentypen von XHydro [Bun07b]

wird durch `unit` repräsentiert.

Bildungsregeln

Wie der gemessene Wert entstanden ist wird durch die Bildungsregeln (`dataType`) beschrieben (siehe Abbildung 2.16). Je nach Art der Bildungsregel (`dataTypeCode`) sind weitere Attribute möglich. Diese werden durch die Code-Listen `aggregation`, `timeStampPosition` und `level` beschrieben.

Qualitätsangaben

Für den Messwert (`dataValueQuality`) sowie für den Zeitstempel (`timeStampQuality`) sind Qualitätsangaben vorgesehen. Die Qualität wird durch die Messunsicherheit bestimmt. Das Messergebnis wird durch ein Intervall der Form *Schätzwert* $\pm$ *Messunsicherheit* repräsentiert. Durch einen Kommentar (`timeStampQualityRemark` und `dataValueQualityRemark`) lässt sich die Qualität noch weiter beschreiben.

[Kd07a]

2.5.4. Das XML Schema von XHydro

Das XML Schema von XHydro wurde manuell aus dem XML Schema Entwurf übernommen. Eine Generierung mit Toolunterstützung konnte die Anforderungen an die Modularität und Wiederverwendbarkeit nicht erfüllen. Um die Anforderungen an die Kompaktheit des Schemas zu erfüllen, wurden die Namen des XML Schema Entwurfs gemäß der Tabelle in Anhang C.1 übersetzt. Die wichtigsten Elemente werden nochmals in Tabelle 2.1 anzeigt.

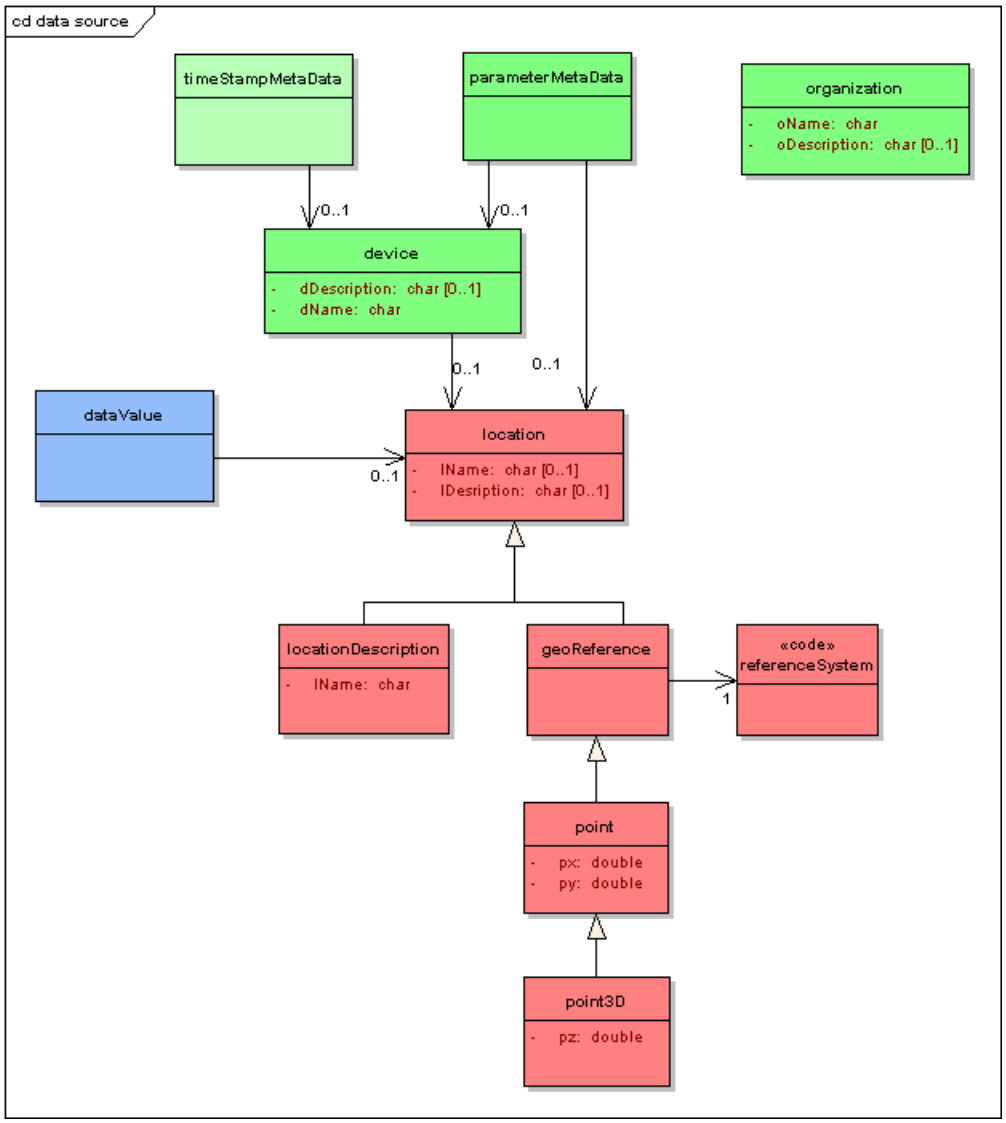


Abbildung 2.14.: Die Beschreibung der Herkunft von Daten von XHydro [Bun07b]

XML Schema Entwurf Name	XML Schema Element Name
data	d
deviceDescription	dd
deviceName	dn
distance	dst
dataType	dt
isochron	iso
locationDescription	ldn
startTimeStamp	sts
timedDataElement	tde
timeStamp	ts

XML Schema Entwurf Name	XML Schema Element Name
timeSeries	tse
timeSeriesList	tsel
timeStampMeasurementInaccuracy	tsmi
timeStampQuality	tsq
tsValue	tsv
dataValue	v
values	vls
dataValueQuality	vq
dataValueQualityRemark	vqr
xDataTypeCode	xdtc
xParameter	xp
xUnit	xu

Tabelle 2.1.: Übersetzung der wichtigsten XML Schema Entwurf Namen in XML Schema Element Namen [Kd07b]

Das XML Schema wird in zwei Versionen angeboten. In der minimalen Version[12] sind die meisten Elemente optional. Dies gewährleistet die Kompaktheit des Dokuments. Die „Archiv"-Version[13] hingegen verpflichtet zur Angabe der meisten Metadaten-Elemente. Wie der Name bereits sagt, dient diese Version der Archivierung. Der Schwerpunkt liegt daher auf der Vollständigkeit und der Korrektheit anstatt auf der Kompaktheit der Daten.

Beide Schema-Versionen verwenden weitestgehend dieselben Module, jedoch mussten an einigen Modulen leichte Veränderungen für die „Archiv"-Version vorgenommen werden. An die Namen dieser Module wird die Bezeichnung *Strict* angehängt.

[Kd07b]

2.5.5. Beispiel

Das Listing in Anhang C.2 zeigt ein Beispiel für die Verwendung des XHydro Schemas. Im folgenden wird näher auf die einzelnen Abschnitte dieses XHydro-Dokuments eingegangen.

Listing 2.27 zeigt das Root-Element von XHydro. Die Attribute dieses Elements *Root-Element*

[12] http://www.xhydro.org/minimal/2007/06 (abgerufen: 16.07.2011)
[13] http://www.xhydro.org/archive/2007/06 (abgerufen: 16.07.2011)

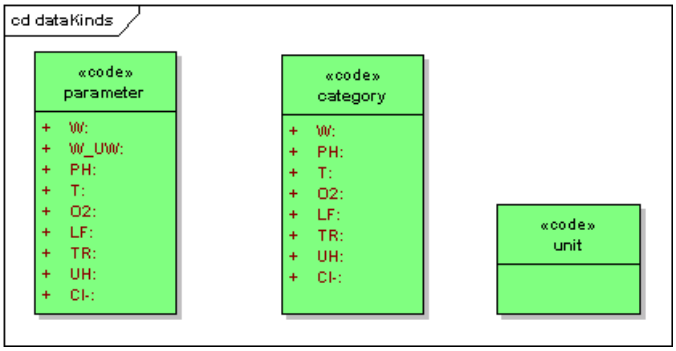


Abbildung 2.15.: Die Messgrößen von XHydro [Bun07b]

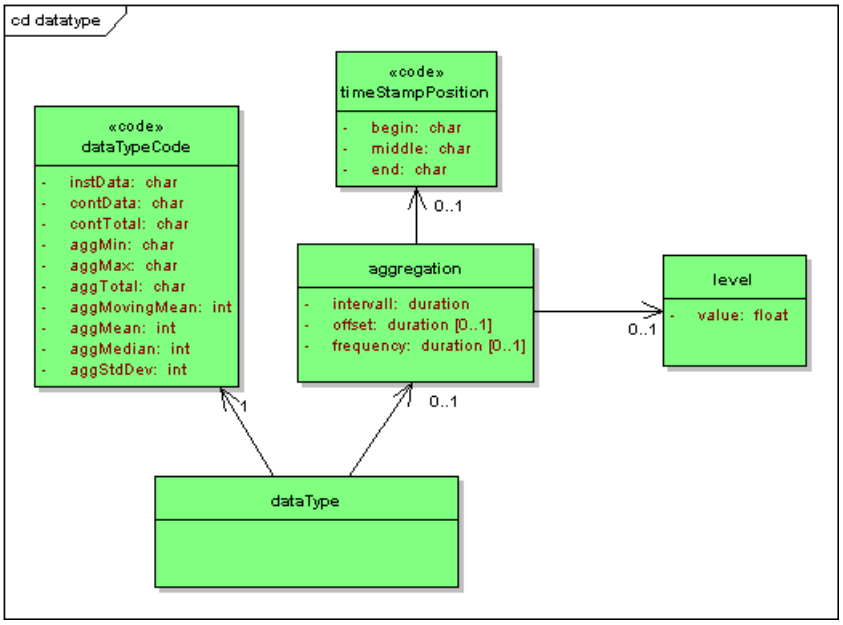


Abbildung 2.16.: Die Bildungsregeln von XHydro [Bun07b]

geben an, welche Namensräume verwendet werden und wo das XHydro Schema im Internet zu finden ist. Zeile 2 gibt den XHydro Namensraum an. In diesem Fall wird das minimale Schema verwendet. Zeile 3 enthält einen herstellerspezifischen Namensraum. Dieser wird in Listing 2.34 verwendet. Zeile 4 enthält den Standard-Namensraum der XML-Schema-Instanz und Zeile 5 und 6 geben den Ort des XHydro Schemas an.

```
2 <tsel xmlns="http://xhydro.org/archive/2007/06"
3  xmlns:d="http://www.disy.net/device"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xsi:schemaLocation="http://xhydro.org/archive/2007/06
6   http://www.xhydro.org/archive/2007/06/XHydroArchive.xsd">
```

Listing 2.27: XHydro-Beispiel: Root-Element [Bun07a]

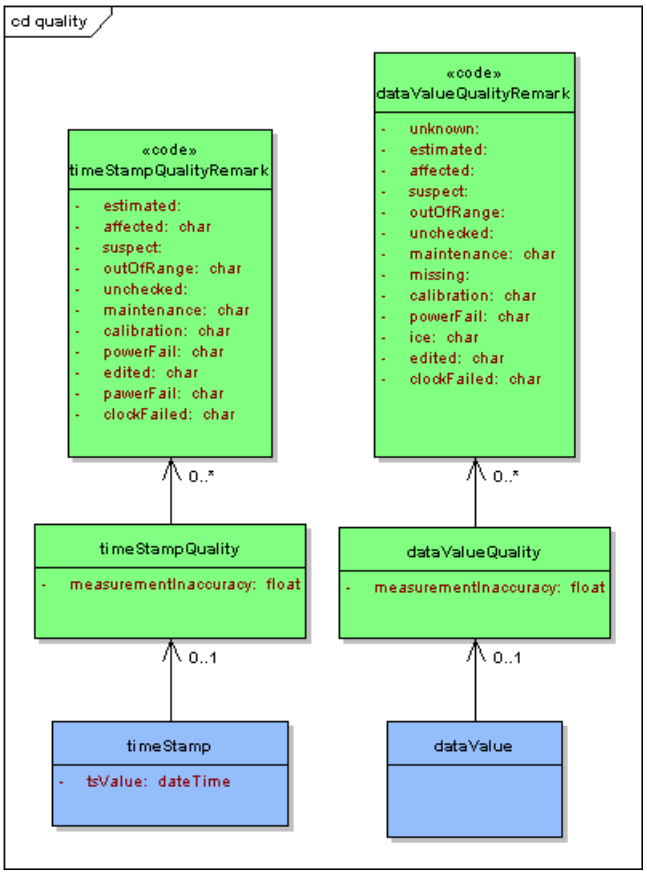


Abbildung 2.17.: Die Qualitätsangaben von XHydro [Bun07b]

Austauschnummern

Listing 2.28 gibt die Austauschnummern an. Wie in Abschnitt 2.5.3 unter „Zeitreihenkern“ bereits erwähnt, dienen diese der einfacheren Zuordnung von importierten Zeitreihen zu internen Zeitreihen. Zeile 12 gibt dabei den Schlüssel und Zeile 13 den Wert an.

```
9  <xids>
10  <xid>
11   <ext />
12   <xk>disy</xk>
13   <xv>4711</xv>
14  </xid>
15 </xids>
```

Listing 2.28: XHydro-Beispiel: Austauschnummern [Bun07a]

Organisation

Die Organisation, zu welcher die Zeitreihe gehört wird in Listing 2.29 angegeben. Zeile 17 gibt den Namen an und Zeile 18 liefert eine Beschreibung der Organisation.

```
16 <org>
17  <on>disy</on>
```

```
18  <od>A company.</od>
19 </org>
```

Listing 2.29: XHydro-Beispiel: Organisation [Bun07a]

Isochronität Die ersten wirklich relevanten Daten für die Zeitreihe werden in Listing 2.30 angegeben. Zeile 20 gibt an, dass es sich bei der folgenden Zeitreihe um eine isochrone Zeitreihe handelt. Die Distanz der Werter dieser Zeitreihe wird in Zeile 21 angegeben. Die Bedeutung des Wertes `P1D` wird später erläutert. Zeile 22-27 beinhalten die Werte für den Start-Zeitstempel. Zeile 24 gibt die Qualität des Zeitstempels (in Form einer Messungenauigkeit) an und Zeile 26 den eigentlichen Zeitstempel.

```
20 <iso>
21  <dst>P1D</dst>
22  <sts>
23   <tsq>
24    <tsmi>1.5E-6</tsmi>
25   </tsq>
26   <tsv>2001-12-31T12:00:00</tsv>
27  </sts>
28 </iso>
```

Listing 2.30: XHydro-Beispiel: Isochronität [Bun07a]

Messgröße Listing 2.31 gibt die Messgröße der Zeitreihe an. Zeile 31 gibt an, dass es sich bei den Daten um den Wasserstand handelt. Zeile 32 gibt einen Kommentar zu der Kategorie an und in Zeile 33 wird die Einheit der Werte angegeben.

```
31 <xp>W</xp>
32 <c>This is a non-standard category code/remark.</c>
33 <xu>m</xu>
```

Listing 2.31: XHydro-Beispiel: Messgröße [Bun07a]

Standort Der Standort wird in Listing 2.32 beschrieben. Der Name wird in Zeile 36 und eine Beschreibung desselbigen in Zeile 37 angegeben.

```
34 <tl>
35  <ldn>
36   <ln>Europe/Germany/Karlsruhe</ln>
37   <ld>Karlsruhe, a german city.</ld>
38  </ldn>
39 </tl>
```

Listing 2.32: XHydro-Beispiel: Standort [Bun07a]

Bildungsregel

Die Messgröße wurde bereits in Listing 2.31 besprochen. Für eine vollständige Beschreibung der Art der Daten ist zusätzlich die Angabe einer Bildungsregel erforderlich. Diese wird in Listing 2.33 angegeben.

Zeile 41 enthält den `dataTypeCode`. In diesem Beispiel handelt es sich bei den Werten um einen Mittelwert. Das Element `it` in Zeile 43 gibt das Intervall der Messung an. Der Abstand der Messungen innerhalb eines Intervalls wird durch das Element `f` (Frequenz) in Zeile 45 bestimmt. Zeile 44 gibt ein Offset an. Es handelt sich dabei um eine Verschiebung der Startzeit einer Messung. Mit diesem Wert lässt sich etwa eine Zeitreihe, die sich an einem Niederschlagstag (beginnt um 7:00 Uhr) orientiert, beschreiben.

Die Darstellung der Zeitabstände orientiert sich an dem XML Datentyp `duration` [W3C04d]. Diese werden in der Form `PnYnMnDTnHnMnS` dargestellt. Das `P` zeigt dabei an, dass es sich um eine Zeitperiode handelt. Es folgen Werte für die Angabe der Jahre (`Y`), der Monate (`M`) und der Tage (`D`). `T` ist ein Trenner zwischen Datums- und Zeitangaben. Dieser wird gefolgt von Werten für die Angabe der Stunden (`H`), der Minuten (`M`) und der Sekunden (`S`). `n` steht dabei für eine natürliche Zahl. Bei den Sekunden ist es jedoch auch möglich, die Hundertstel durch einen Punkt getrennt anzuzeigen.

Zeile 46 beschreibt die Art des Zeitstempels. In diesem Fall beschreibt der Zeitstempel den Beginn des Intervalls. Zeile 47 gibt an, dass es sich um eine Zeitreihe handelt, die die Häufigkeit von Überschreitungen eines bestimmten Wertes darstellt.

```
40 <dt>
41  <xdtc>aggMean</xdtc>
42  <ag>
43   <it>P1D</it>
44   <ot>PT15M</ot>
45   <f>PT1M</f>
46   <xtsp>begin</xtsp>
47   <l>1.5</l>
48  </ag>
49 </dt>
```

Listing 2.33: XHydro-Beispiel: Bildungsregel [Bun07a]

Beschreibung der Herkunft der Daten

Listing 2.34 beschreibt die Herkunft der Daten. Die Zeilen 51 - 53 verwenden den in Listing 2.27 (Zeile 3) eingeführten Namensraum. Weiterhin geben die folgenden Zeilen einen Gerätenamen (Zeile 54), eine Gerätebeschreibung (Zeile 55) und einen Ort (Zeilen 57 - 60) an. Der Ort wird in Zeile 58 durch einen Namen und in Zeile 59 durch eine Beschreibung angegeben.

```
50 <pd>
51  <ext>
52   <d:serial>ABCDEFG</d:serial>
53  </ext>
54  <dn>dd</dn>
55  <dd>A dummy disy device.</dd>
56  <dl>
57   <ldn>
58    <ln>Europe/Germany/Karlsruhe</ln>
59    <ld>Karlsruhe, a german city.</ld>
60   </ldn>
61  </dl>
62 </pd>
```

Listing 2.34: XHydro-Beispiel: Beschreibung der Herkunft der Daten [Bun07a]

Messwert-Qualität

In Listing 2.35 wird die Messwert-Qualität angegeben. Dies geschieht durch die Angabe einer Messungenauigkeit in Zeile 64 und einer Anmerkung in den Zeilen 65 - 69. Die Angaben gelten global und können lokal überschrieben werden (siehe Listing 2.38).

```
63 <vq>
64  <vmi>5E-3</vmi>
65  <vqr>
66   It is a quite imprecise device, isn't it? This quality
67   remark demonstrates that free-text remarks are possible,
68   too.
69  </vqr>
70 </vq>
```

Listing 2.35: XHydro-Beispiel: Messwert-Qualität [Bun07a]

Beschreibung der Herkunft der Zeitstempel

Die Beschreibung der Herkunft des Zeitstempels wird in Listing 2.36 angezeigt. Dieser wird in Zeile 75 durch einen Namen und in Zeile 76 durch eine Beschreibung angegeben.

```
74 <tsd>
75  <dn>ddts</dn>
76  <dd>A dummy dis device to measure time.</dd>
77 </tsd>
```

Listing 2.36: XHydro-Beispiel: Beschreibung der Herkunft des Zeitstempels [Bun07a]

Zeitstempel-Qualität

Die Zeitstempel-Qualität wird durch die Angebe einer Messungenauigkeit in

Zeile 79 in Listing 2.37 angegeben. Weiterhin wird in den Zeilen 80 - 83 eine herstellerspezifische Code-Liste verwendet.

```
78 <tsq>
79  <tsmi>1.5E-6</tsmi>
80  <tsqr codeList="disy1" codeListAgency="disy"
81   codeListVersion="1.0">
82   ownCode
83  </tsqr>
84 </tsq>
```

Listing 2.37: XHydro-Beispiel: Zeitstempel-Qualität [Bun07a]

Daten

Die eigentlichen Daten werden in Listing 2.38 dargestellt. Insgesamt werden drei verschiedene Werte angegeben, welche alle auf eine andere Weise dargestellt werden.

Die Zeilen 90 - 103 stellen einen Wert (Zeile 102) zusammen mit Informationen bezüglich der Qualität (Zeilen 91 - 94) und des Standorts (Zeilen 95 - 101) dar. Wie zu sehen ist, wurde die Qualitätsangabe aus Listing 2.35 für diesen Wert überschrieben. Der Standort wird diesmal durch einen Punkt beschrieben. Dazu wird eine Referenz auf die `ReferenceSystem` Code-Liste (Zeile 97), sowie zwei Koordinaten x (Zeile 98) und y (Zeile 99) angegeben.

Der zweite Wert (Zeilen 104 - 106) wurde ohne jegliche zusätzliche Angaben eingetragen.

Der dritte Wert (Zeilen 107 - 112) wurde als fehlend angegeben. Dies geschieht durch den Qualitäts-Eintrag `missing` in Zeile 109. Als Wert wird ein leerer `d-anyValue` Wert (Zeile 111) verwendet.

```
90  <v>
91   <vq>
92    <vmi>6E-4</vmi>
93    <xvqr>affected</xvqr>
94   </vq>
95   <vl>
96    <pt>
97     <xrs>32632</xrs>
98     <px>5.0</px>
99     <py>6.0</py>
100   </pt>
101  </vl>
102  <vf>4.5</vf>
103 </v>
104 <v>
105  <vf>4.6</vf>
```

```
106 </v>
107 <v>
108  <vq>
109   <xvqr>missing</xvqr>
110  </vq>
111  <va xsi:nil="true" />
112 </v>
```

Listing 2.38: XHydro-Beispiel: Daten [Bun07a]

3. Anforderungen an das System

In diesem Kapitel werden die wichtigsten, realisierten Anforderungen an PegelSuite vorgestellt. Dazu wird in Abschnitt 3.1 eine Systemübersicht anhand eines UseCase-Diagramms beschrieben. Im Anschluss werden dann die Anforderungen (siehe Abschnitt 3.2), unterteilt in die einzelnen Bereiche funktionale Anforderungen (siehe Abschnitt 3.2.1), technische Anforderungen (siehe Abschnitt 3.2.2), Qualitätsanforderungen (siehe Abschnitt 3.2.3) und Anforderungen an den Entwicklungsprozess (siehe Abschnitt 3.2.4), beschrieben. Das komplette Pflichtenheft des Projekts ist in Anhang D zu finden.

3.1. Systemübersicht

Das Produkt *PegelSuite* ist ein Web Service orientiertes System zur Verarbeitung von Pegeldaten. Es umfasst das Empfangen bzw. Abfragen von Daten der einzelnen Pegel, sowie deren Verarbeitung. Dazu gehören auch das Vergleichen der Sensoren einer Messstelle, die Plausibilisierung eines Gewässerabschnitts, sowie das Versenden von Alarmen. Diese Grundfunktionen entsprechen den einzelnen Workflows, welche bereits in Abschnitt 1.4 erläutert wurden. Anhand von Abbildung 3.1 werden die Anwendungsfälle, des Systems nun nochmals dargestellt.

PegelSuite sammelt die Pegeldaten von den einzelnen Pegeln. Dies kann im Abruf- oder Push-Betrieb möglich sein. Welcher der beiden Betriebsarten vorliegt, kann je Pegel variieren. Tritt bei dieser Aktion ein Fehler auf, so versendet PegelSuite einen Alarm. Ankommende Sensordaten müssen verglichen und anschließend gespeichert werden. Bei Fehlern wird ebenfalls ein Alarm gesendet. Dies ist ebenfalls bei fehlgeschlagener Plausibilisierung der Pegeldaten der Fall. Sollte die Plausibilisierung erfolgreich sein, werden die Daten gespeichert. Alle diese Aufgaben übernimmt die PegelSuite, jedoch müssen die versendeten Alarme von einem Sachbearbeiter manuell bearbeitet werden.

3.2. Anforderungen

Die im Folgenden vorgestellten Anforderungen aus dem Pflichtenheft wurden nummeriert um auf diese verweisen zu können. Die Nummerierung hat folgende Form:

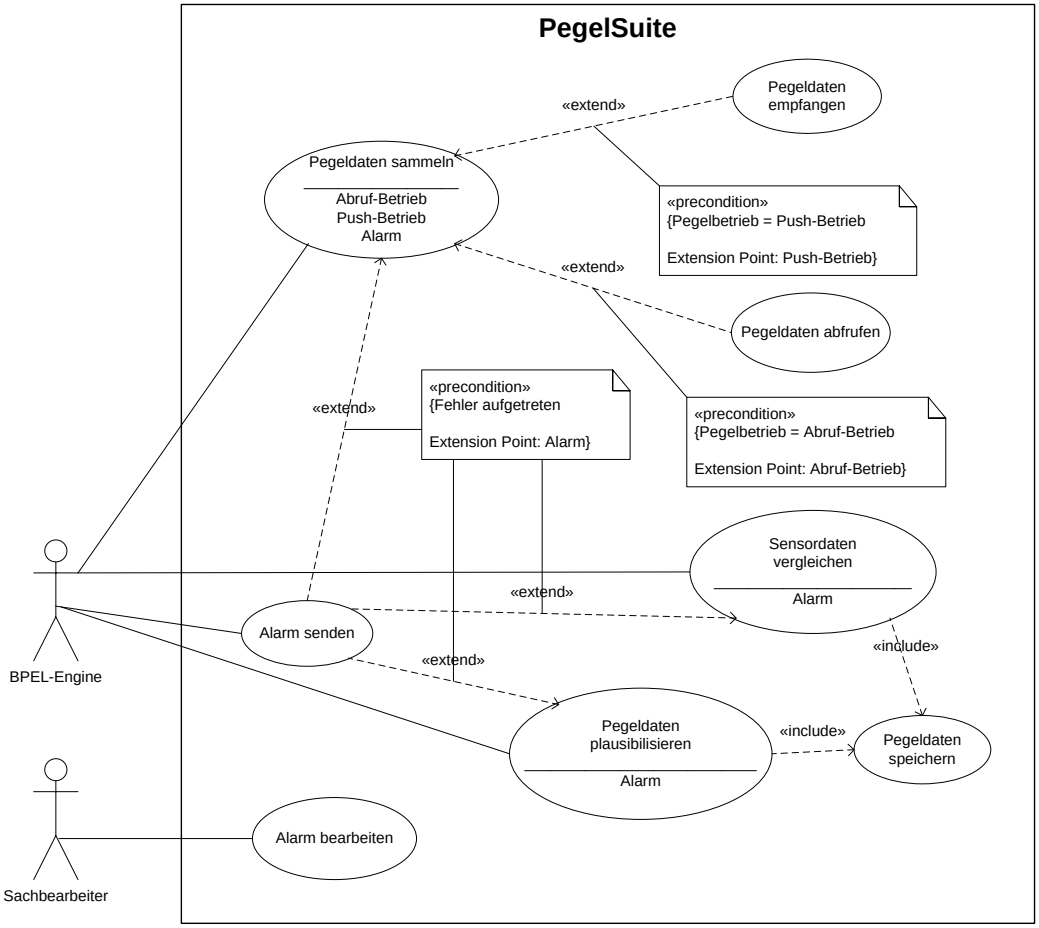


Abbildung 3.1.: Systemübersicht *PegelSuite*

Anforderungstyp-Nummer{**.Zwischennummer**}

Der **Anforderungstyp** wird in Kurzform angegeben. Bei **Nummer** und **Zwischennummer** handelt es sich um eine fortlaufende Nummerierung der Anforderungen. Die Angabe einer **Zwischennummer** bedeutet, dass neue Anforderungen, zwischen zwei Anforderungen eingefügt wurden. Beim Einfügen von Anforderungen zwischen zwei Anforderungen, welche bereits eine **Zwischennummer** besitzen, ist eine weitere **Zwischennummer** angehängt. Diese Art der Nummerierung ist nötig, um die Nummerierung der bereits vorhandenen Anforderungen unverändert zu lassen. Anforderungen, welche nicht weiter gültig sind, werden durchgestrichen dargestellt.

Die Anforderungen unterliegen verschiedenen Verbindlichkeiten. Diese lassen sich anhand einiger *hervorgehobener* Schlagworte (*muss (nicht) > soll (nicht) > sollte (nicht) > kann (nicht)*) in den Anforderungen bestimmen.

Für die funktionalen Anforderungen (siehe Abschnitt 3.2.1) werden innerhalb einer Verbindlichkeitsstufe die Anforderungen durch verschiedene Prioritäten (0

(unwichtig) - 5 (wichtig)) nochmals unterteilt. Die Priorität wird *hervorgehoben* und in Klammern hinter der Anforderung genannt.

3.2.1. Funktionale Anforderungen

Dieser Abschnitt beschreibt die funktionalen Anforderungen an das System. Diese beschrieben die Funktion des Systems.

Workflow 1: Datenabruf und -empfang

F-1 Die Daten *müssen* von den Pegeln abgerufen werden können. *(Priorität: 5)*
F-2 Die Daten *müssen* im Push-Betrieb empfangen werden können. *(Priorität: 5)*

Die Anforderungen **F-1** und **F-2** sind essentiell für den Betrieb. Bevor Daten verarbeitet werden können, müssen sie schließlich erst einmal auf irgendeine Weise beschafft werden. Im Fall von PegelSuite kann dies durch Pull- und Push-Betrieb geschehen.

F-3 Der Push-Betrieb *muss* dem Pull-Betrieb vorgezogen werden. *(Priorität: 5)*

Das der Push-Betrieb dem Pull-Betrieb vorgezogen wird (**F-3**), hat den Vorteil, dass PegelSuite sich weniger mit dem Abfragen der Daten befassen muss. Dadurch wird die Performance gesteigert. Weiterhin wird durch das Wegfallen der Anfrage die Netzwerkkommunikation verringert.

F-4 Das System *muss* nicht ankommende Daten im Push-Betrieb, im Pull-Betrieb abfragen. *(Priorität: 5)*
F-5 Vor dem Wechsel in den Pull-Betrieb *muss* eine variable Zeit (Standard: 15 Minuten) auf die fehlenden Daten gewartet werden. *(Priorität: 4)*

Natürlich kann es vorkommen, das Daten im Push-Betrieb nicht ankommen. In diesem Fall sollen diese nach einer variablen Zeit (**F-5**), im Pull-Betrieb abgefragt werden (**F-4**).

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 2: Sensorvergleich

F-8 Falls Daten mehrerer Sensoren vorliegen, *müssen* diese verglichen werden. *(Priorität: 5)*

F-9 Zu den Daten *muss* ein Qualitätsmerkmal (siehe Abschnitt D.7.1) erstellt werden. *(Priorität: 5)*

Um Fehlmessungen zu bestimmen, kann es sein, dass ein Pegel mehrere Sensoren besitzt. Diese müssen verglichen werden (**F-8**). Anforderung **F-9** besagt, dass dies durch ein Qualitätsmerkmal passieren muss. Es handelt sich bei dem Qualitätsmerkmal um den Mittelwert der einzelnen Pegel.

F-10 Die Daten aller Sensoren *müssen* in der Datenbank gespeichert werden. *(Priorität: 5)*

F-11 Das Qualitätsmerkmal *muss* in der Datenbank gespeichert werden. *(Priorität: 5)*

Natürlich müssen die Daten auch archiviert werden. Dies geschieht durch Speichern der Pegeldaten (**F-10**) und des Qualitätsmerkmals (**F-11**) in einer Datenbank.

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 3: Plausibilisierung

F-13 Die Daten eines gesamten Gewässerabschnitts *müssen* plausibilisiert werden (siehe Abschnitt D.7.2). *(Priorität: 5)*

F-14 ~~Die Qualität der plausibilisierten Daten (gleitender Mittelwert; temporär) *muss* berechnet werden. *(Priorität: 5)*~~

Anforderung **F-13** besagt, dass die Daten eines gesamten Gewässerabschnitts plausibilisiert werden müssen. Dies geschieht durch die Fuzzylogik der BfG, wodurch Anforderung **F-14** von PegelSuite nicht beachtet werden muss.

F-15 Zur Plausibilisierung *muss* der Pegelwert des höchst priorisierten, funktionierenden Sensors genommen werden. *(Priorität: 5)*

Laut **F-15** muss die Plausibilisierung nur für den höchst priorisierten, funktionierenden Sensor durchgeführt werden. Dies ist der Sensor mit der kleinsten Sensornummer.

F-16 Die plausibilisierten Daten *müssen* in der Datenbank gespeichert werden. *(Priorität: 5)*

F-16.1 Es *müssen* nur die Daten des an der Plausibilisierung beteiligten Sensors als plausibilisierte Daten übernommen werden. *(Priorität: 5)*

Nach einer bestandenen Plausibilisierung werden die Daten nochmals in der Datenbank gespeichert (**F-16**). Dies geschieht nun jedoch in einer anderen Tabelle als für die Rohdaten, da jetzt nurnoch dieDAten der an der Plausibilisierung beteiligten Sensoren übernommen werden (**F-16.1**).

F-17 ~~Der Workflow *soll* automatisch nach einer bestimmten Zeitspanne (variabel) gestartet werden. *(Priorität: 3)*~~

F-17.1 Der Workflow *soll* von Hand aufrufbar sein. *(Priorität: 4)*

F-17.2 Der Workflow *soll* gestartet werden, sobald genügend Daten für die Plausibilisierung vorliegen. *(Priorität: 3)*

Anforderung **F-17** besagte, dass die Plausibilisierung nach einer bestimmten Zeitspanne automatisch gestartet werden soll. Diese wurde allerdings während der Entwicklung verworfen und dafür die Anforderungen **F-17.1** und **F-17.2** eingefügt. Diese besagen, das die Plausibilisierung manuell (**F-17.1**) oder sobald genügend Daten vorhanden sind (**F-17.2**) angestoßen werden kann.

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 4: Alarmmanagement

F-18 Das System *muss* Alarme per E-Mail versenden können. *(Priorität: 5)*

Wie bereits erwähnt müssen die Fehler, in den vorangegangenen Workflows als Alarm versendet werden. Dies ist im Moment per E-Mail möglich, wie es in **F-18** gefordert wird.

F-20 ~~Es *sollen nicht* alle zuständigen Stellen gleichzeitig benachrichtigt werden. *(Priorität: 4)*~~

F-20.1 Es *sollen* alle Personen innerhalb einer Eskalationsstufe gleichzeitig benachrichtigt werden. *(Priorität: 4)*

F-20.2 Eine Person *soll* im Alarmfall über alle angegebenen Kontaktadressen gleichzeitig benachrichtigt werden. *(Priorität: 3)*

Die Alarme sollen dabei aber nicht an alle Personen gleichzeitig gesendet werden (**F-20**). Diese Anforderung wurde durch die Anforderungen **F-20.1** und **F-20.2** ersetzt, die ein genaueres System beschrieben. Dieses besteht aus mehreren Eskalationsstufen, wobei alle Personen einer Eskalationsstufe gleichzeitig (**F-20.1**) über alle ihre angegeben Kontaktadressen (**F-20.2**) benachrichtigt werden sollen.

F-21 Es *soll* erst nach Ablauf einer variablen Zeit die nächste Stelle benachrichtigt werden. *(Priorität: 4)*

F-22 Die Zeit bis zur nächsten Eskalationsstufe *kann* pro Person einstellbar sein. *(Priorität: 0)*

F-22.1 Vor dem Wechsel in die nächste Eskalationsstufe *kann* der Alarm, beim Ausbleiben von Quittierungen, mehrmals wiederholt werden. *(Priorität: 0)*

Die Eskalationsstufe wird dabei nach Ablauf einer variablen Zeit (**F-21**), welche pro Person einstellbar ist (**F-22**) gewechselt. Allerdings kann nach dieser Zeit (**F-21**) statt eines Wechsels der Eskalationsstufe auch eine Wiederholung des Alarms auf der selben Eskalationsstufe gesendet werden (**F-22.1**).

F-23 Die Bearbeitung eines Alarms *muss* quittiert werden. *(Priorität: 3)*

Gemeint ist mit Anforderung **F-23**, dass ein Alarm, sobald er von einer Person bearbeitet wird, sein Alarmstatus auf „wird bearbeitet“ gesetzt wird. So kann nachgeprüft werden, ob sich bereits jemand um den Alarm kümmert.

F-23.1 Die Zuständigkeit für einen Alarm *muss* pro Pegel und Alarmtyp konfigurierbar sein. *(Priorität: 3)*

Wer für einen Alarm zuständig ist sollte ebenfalls konfigurierbar sein. Dies muss laut **F-23.1** pro Pegel und Alarmtyp möglich sein.

Workflow 1 Für „Workflow 1: Datenabruf und -empfang“ wird ein Alarm gesendet …

F-24 … wenn der Abruf der Daten gescheitert ist. *(Priorität: 5)*

F-25 Die Anzahl der Versuche, bevor ein Alarm gesendet wird *soll* variabel (Standard: 1 mal) sein. *(Priorität: 3)*

F-26 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen werden. *(Priorität: 3)*

Beim Abruf von Daten kann es vorkommen, dass der Pegel z.B. nicht erreichbar ist und dadurch der Abruf der Daten scheitert. In diesem Fall muss ein Alarm gesendet werden (**F-24**). Dies geschieht allerdings erst nach einer variablen Anzahl von Versuchen (**F-25**). Kommen in einem Alarmzustand Daten an, so sollen diese weiterhin angenommen werden (**F-26**).

Für „Workflow 2: Sensorvergleich“ wird ein Alarm gesendet … *Workflow 2*

F-27 …wenn Sensoren fehlerhaft sind. *(Priorität: 5)*

F-28 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen und gespeichert werden. *(Priorität: 3)*

Sind Sensoren fehlerhaft, soll ein Alarm gesendet werden (**F-27**). Dies ist der Fall, wenn der betroffene Sensor keine Daten mehr sendet. Sollten im Alarmzustand Daten ankommen, dann sollen diese weiterhin entgegen genommen werden (**F-28**).

F-29 …wenn die Abweichung der Daten zum gleitenden Mittelwert zu groß ist. *(Priorität: 4)*

F-30 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen und gespeichert werden. *(Priorität: 2)*

F-31 Die erlaubte Abweichung *soll* variabel (Standard: 2 - 3 cm) sein. *(Priorität: 2)*

Weiterhin soll in Workflow 2 ein Alarm gesendet werden, wenn die Abweichung zum Qualitätsmerkmal zu groß ist (**F-29**). Auch in diesem Fall sollen ankommende Daten weiterhin angenommen werden (**F-30**). Die erlaubte Abweichung vom Qualitätsmerkmal soll variabel gehalten werden (**F-31**).

Für „Workflow 3: Plausibilisierung“ wird ein Alarm gesendet … *Workflow 3*

F-35.1.1 …wenn die Daten nicht plausibilisiert werden konnten. *(Priorität: 4)*

F-35.1.2 Die derzeitige (für diesen Zeitraum und Pegel) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*

F-35.2 Abgebrochene Plausibilisierungen *müssen* von Hand nachgetragen oder „Workflow 3: Plausibilisierung“ manuell gestartet werden. *(Priorität: 4)*

In dem Fall das eine Plausibilisierung fehl schlägt, wird ein Alarm gesendet (**F-35.1.1**). Das Abbrechen der Plausibilisierung (**F-35.1.2**) kann nicht von Pegel-Suite aus geschehen, sondern muss von der Fuzzy Logic vorgenommen werden. Nachdem Abbrechen muss die Plausibilisierung manuell nachgetragen oder gestartet werden (**F-35.2**).

F-36 …wenn zur Plausibilisierung benötigte Daten fehlen. *(Priorität: 5)*

F-37.1.1 Die derzeitige (für diesen Zeitraum und Pegel) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*

F-37.2 Abgebrochene Plausibilisierungen *müssen* von Hand nachgetragen oder „Workflow 3: Plausibilisierung“ manuell gestartet werden. *(Priorität: 4)*

Weiterhin versendet Workflow 3 einen Alarm, wenn Daten für die Plausibilisierung fehlen (**F-36**). Auch in diesem Fall muss die Ausführung der Plausibilisierung abgebrochen (**F-37.1.1**) werden und kann später manuell nachgetragen oder gestartet werden (**F-37.2**).

3.2.2. Technische Anforderungen

Dieser Abschnitt beschreibt die technischen Rahmenbedingungen an PegelSuite. Im Folgenden werden die allgemeinen technischen Anforderungen beschrieben und anschließend in Unterabschnitten, die speziell auf den jeweiligen Bereich zugeschnittene Anforderungen.

T-1.1 Das System *muss* frei verfügbare Software verwenden.

Die verwendetes Software, auf welcher, das System basiert, muss laut Anforderung **T-1.1** frei verfügbar sein. Dies betrifft Entscheidungen bezüglich des zu verwendenden Web Servers und der zu verwendenden BPEL-Engine etc.

T-1 Das System *muss* auf Web Services basieren.
T-2 Das System *soll* SOAP über HTTP verwenden.
T-3 Die Web Services *sollen* in Java implementiert werden.

Weiterhin muss das System auf Web Services basieren (**T-1**), welche zur Übertragung SOAP über HTTP verwenden (**T-2**; siehe Abschnitt 2.3.4). Als Programmiersprache für die Web Services und auch den Rest des Systems wurde Java gewählt (**T-3**).

T-4 Die Web Services *müssen* von einer BPEL-Engine orchestriert werden.
T-5 Als BPEL-Engine *sollte* Apache ODE verwendet werden.

Nicht in Java geschrieben wird allerdings die Orchestrierung der Web Services. Diese muss durch BPEL geschehen (**T-4**; siehe Abschnitt 2.4).

Als frei verfügbare BPEL-Engine wird Apache ODE verwendet (**T-5**).

T-8 Das System *muss* im unbeaufsichtigten Betrieb laufen.
T-9 Das System *soll* rund um die Uhr erreichbar sein.

Laut **T-8** muss das System im unbeaufsichtigten Zustand laufen und dabei Rund um die Uhr erreichbar sein (**T-9**). Da das System die manuelle Arbeit der Plausibilisierung der Daten, den Sachbearbeitern abnehmen soll, ist es nur sinnvoll, dass das System autonom läuft und nur bei Alarmen das Mitwirken von Personen erfordert. Da das System im Push-Betrieb Daten empfangen soll (siehe **F-2**) muss es im Betrieb rund um die Uhr für die Pegel erreichbar sein.

T-10 Das System *muss* sich an bestehende Standards halten.

T-10 schreibt vor, dass das System Standardkonform sein muss. Dadurch werden Standards wie XHydro, BPEL etc. obligatorisch.

T-11 Die einzelnen Komponenten des Systems *sollen* leicht austauschbar sein.

Austauschbare Komponenten (**T-11**) sind z.B. durch die Verwendung von Web Services mit einer standardisierten Schnittstelle gegeben. Auch bei den anderen Komponenten sollte auf eine lose Kopplung geachtet werden.

Datenbank

T-12 Das Datenbanksystem *soll* MySQL sein.
T-13 Die Datenbank *soll* als Storage-Engine InnoDB verwenden.
T-14 Das Datenbank Encoding *muss* UTF-8 sein.

Als Datenbanksystem wird für PegelSuite MySQL verwendet (**T-12**). Die Storage-Engine soll dabei laut **T-13** InnoDB sein. Diese gewährleistet Transaktionssicherheit und referenzielle Integrität. Als Encoding verwendet die Datenbank UFT-8 (**T-14**).

XHydro

T-15 Innerhalb des Systems *soll* XHydro als Datenaustauschformat verwendet werden.

Anforderung **T-15** schreibt vor, dass XHydro als Datenaustauschformat innerhalb des Systems verwendet werden soll. Verwendet wurde XHydro allerdings nur für die Schnittstellen zwischen Pegel und PegelSuite.

3.2.3. Qualitätsanforderungen

Die folgenden Qualitätsanforderungen orientieren sich am Industriestandard. Diese wird als Standardqualität definiert.

#	Qualität	sehr gut	gut	normal
	Funktionalität			
Q-1	Angemessenheit		✘	
Q-2	Richtigkeit		✘	
Q-3	Interoperabilität	✘		
Q-4	Ordnungsmäßigkeit		✘	
Q-5	Sicherheit			✘

#	Qualität	sehr gut	gut	normal
Zuverlässigkeit				
Q-6	Reife			✘
Q-7	Fehlertoleranz			✘
Q-8	Wiederherstellbarkeit			✘
Benutzbarkeit				
Q-9	Verständlichkeit	✘		
Q-10	Erlernbarkeit	✘		
Q-11	Bedienbarkeit	✘		
Effizienz				
Q-12	Zeitverhalten	✘		
Q-13	Verbrauchsverhalten			✘
Änderbarkeit				
Q-14	Analysierbarkeit		✘	
Q-15	Modifizierbarkeit	✘		
Q-16	Stabilität			✘
Q-17	Prüfbarkeit			✘
Übertragbarkeit				
Q-18	Anpassbarkeit		✘	
Q-19	Installierbarkeit			✘
Q-20	Konformität	✘		
Q-21	Austauschbarkeit	✘		

Tabelle 3.1.: Qualitätsanforderungen

Funktionalität Die *Funktionalität* wird durch die Qualitätsmerkmale Angemessenheit (**Q-1**), Richtigkeit (**Q-2**), Interoperabilität (**Q-3**), Ordnungsmäßigkeit (**Q-4**) und Sicherheit (**Q-5**) beschrieben. Dabei wird sehr viel Wert auf die Interoperabilität von PegelSuite gelegt, da die Kommunikation zwischen Pegel und PegelSuite essentiell für die Funktion von PegelSuite ist. Eine gute Qualität ist für die Merkmale Angemessenheit, Richtigkeit und Ordnungsmäßigkeit nötig. Das System soll auf angemessene Weise die Funktionen des Systems korrekt und ordnungsgemäß ausführen. In dem in der Diplomarbeit zu erstellenden Prototypen wurde die Sicherheit als nicht so wichtig eingestuft.

Zuverlässigkeit Für die *Zuverlässigkeit* gelten die Qualitätsmerkmale Reife (**Q-6**), Fehlertoleranz (**Q-7**) und Wiederherstellbarkeit (**Q-8**). Diese wurden alle auf dem Niveau des Industriestandards für einen Prototypen angesiedelt.

Benutzbarkeit Die Qualitätsmerkmale für die *Benutzbarkeit* von PegelSuite sind Verständlichkeit

(**Q-9**), Erlernbarkeit (**Q-10**) und Bedienbarkeit (**Q-11**). Durch die Verwendung von Javadoc und einer guten Kommentierung des Codes ist dieser für Außenstehende leicht zu erlernen und zu verstehen. Für den Anwender gibt es eigentlich nur zu Verstehen und zu Erlernen, wie man PegelSuite konfiguriert und startet. Dies ist leider in der Prototypenversion nur durch Manipulation von Variablen im Code möglich. Auch das Beschaffen weiterer Informationen zu Alarmen erfordert die Kenntnis von SQL. Allein der Statuswechsel von Alarmen ist durch einen Link über ein REST-Interface möglich. Die Wiederherstellbarkeit bei Absturz des Systems wurde für den Prototypen auch als nicht ganz so wichtig angesehen.

Effizienz

Qualitätsmerkmale für die *Effizienz* von PegelSuite sind das Zeitverhalten (**Q-12**) und das Verbrauchsverhalten (**Q-12**). Das Zeitverhalten von PegelSuite sollte möglichst hochperformant sein. Das Verbrauchsverhalten entspricht dem Industriestandard. Durch die Menge der zu speichernden Daten ist es z.B. nicht möglich einen geringen Verbrauch an Speicherkapazität zu erreichen.

Änderbarkeit

Die *Änderbarkeit* des Systems wird durch die Qualitätsmerkmale Analysierbarkeit (**Q-14**), Modifizierbarkeit (**Q-15**), Stabilität (**Q-16**) und Prüfbarkeit (**Q-17**) gewährleistet. Insbesondere die Modifizierbarkeit von PegelSuite ist dabei von großem Interesse. Gut analysieren lässt sich das Programm durch die Einhaltung von Konventionen, wie z.B. Einrückung des Codes und Namensgebung der Variablen. Weiterhin ist auch die Kommentierung des Codes wichtig um dies zu erreichen. Die Stabilität und Prüfbarkeit des Prototyps von PegelSuite sind hingegen weniger wichtig.

Anpassbarkeit

Anpassbarkeit (**Q-18**), Installierbarkeit (**Q-19**), Konformität (**Q-20**) und Austauschbarkeit (**Q-21**) sind die Qualitätsmerkmale für die *Übertragbarkeit* des Systems. Die wichtigsten Qualitätsmerkmale für PegelSuite sind Konformität und Austauschbarkeit. Bei der Implementation ist daher auch die Verwendung von Standards und loser Kopplung geachtet worden. PegelSuite sollte weiterhin an gegebene Umstände anpassbar sein. Wie einfach sich PegelSuite installieren lässt wurde nicht als so wichtig angesehen.

3.2.4. Anforderungen an den Entwicklungsprozess

E-1 Das zu modellierende Datenbankschema *muss* sich an XHydro orientieren.

Anforderung **E-1** besagt, dass sich das Datenbankschema (siehe Abschnitt 4.2) an XHydro (siehe Abschnitt 2.5) orientieren muss. Gemeint ist, dass alle Informationen, die XHydro liefern kann, auch in der Datenbank abgelegt werden müssen.

E-2 Es *soll* eine datenbankunabhängige Java-Schnittstelle (etwa JDBC) verwendet werden.

Die Schnittstelle für den Zugriff auf die Datenbank soll dabei laut **E-2** datenbankunabhängig sein.

4. Entwurf und Implementation

Dieses Kapitel beschreibt den Entwurf und die Implementierung des System. Abschnitt 4.1 beschreibt die Entwurfsentscheidungen, die aufgrund der Anforderungen getroffen wurden. In den folgenden Abschnitten wird dann der Entwurf der Datenbank (Abschnitt 4.2), der BPEL-Workflows (Abschnitt 4.3) und der Java-Komponenten (Abschnitt 4.4) vorgestellt. Anschließend werden in Abschnitt 4.5 die Probleme bei der Implementierung beschrieben.

4.1. Entwurfsentscheidungen

Die Entwurfsentscheidungen in diesem Abschnitt beziehen sich auf die Anforderungen in Abschnitt 3.2. Diese werden im Folgenden anhand ihrer Nummer benannt.

Eine der wichtigsten Anforderungen war die Verwendung von WS-BPEL und anderen Standards (**T-4**, **T-10**). Daher wurde beschlossen möglichst viel in BPEL zu implementieren. Der direkte Datenbankzugriff mit BPEL kann jedoch nur einzeilige Antworten auf eine Abfrage liefern, daher macht es wenig Sinn diesen zu verwenden. Zudem war auch die Dokumentation, wie diese einzubinden ist, veraltet. Der Datenbankzugriff wurde daher als Java-Komponente realisiert. Alle in BPEL nicht realisierbaren Aktionen wurden in Java implementiert. Die Kommunikation zwischen BPEL und Java erfolgt dabei durch Web Services.

Nicht nur im Fall der Datenbank war die Dokumentation veraltet. WS-Security (**T-19**) konnte aus diesem Grund ebenfalls nicht realisiert werden.

Auch die Austauschbarkeit der Komponenten war eine Anforderung, die den Entwurf beeinflusste (**T-11**). Durch die Verwendung von Web Services (**T-1**, **T-2**), ist dies bereits an den Schnittstellen nach außen erfüllt. Weitere Komponenten wurden ebenfalls entsprechend Anforderung **T-11** entworfen. So lässt sich beispielsweise die Datenbank durch die Verwendung von JDBC (**E-2**) leicht austauschen.

Der Standard XHydro sollte eigentlich laut Anforderung **T-15** auch innerhalb des Systems verwendet werden. Da das Format allerdings zu komplex ist, wurde es

nur für die Kommunikation zwischen PegelSuite und Pegel verwendet. Allerdings spielte XHydro auch bei dem Entwurf der Datenbank (**E-1**) eine Rolle. In der Datenbank sollten alle Daten gespeichert werden können, welche XHydro liefern kann.

Eigentlich sollte das System zudem eine hohe Performanz vorweisen (**Q-12**). Diese Anforderung konnte allerdings aufgrund anderer Entwurfsentscheidungen (z.B. der Datenbankzugriff über Java) und der daraus resultierenden Probleme (z.B. überflüssige Kommunikation durch viele Web Services) nicht erfüllt werden (siehe Abschnitt 4.5).

4.2. Datenbank

Ein wichtiger Teil des System ist eine gut modellierte Datenbank. Dafür müssen zuerst die zu speichernden Daten identifiziert werden (Abschnitt 4.2.1). Diese werden dann in einem konzeptuellen Modell in Beziehung zu einander gesetzt (Abschnitt 4.2.2) und schließlich ins logische Modell überführt (Abschnitt 4.2.3). Letztendlich wird dieses in die entsprechenden SQL-Befehle zum Erstellen der Tabellen in der Datenbank verwendet (Abschnitt 4.2.4).

4.2.1. Datenbankentitäten

Als Daten für die Datenbank kommen neben den Daten, die ein XHydro-Dokument liefern kann, auch verwaltungstechnische Daten in Frage. Alle diese Daten werden im folgenden identifiziert und zusammengetragen.

- Pegel-Stammdaten
  - Standort-ID
  - Standort-Name
  - IP-Adresse
  - Pfad zu `getTimeSeries` Web Service
  - Austauschformat
  - PNP
  - Stromkilometer
  - Intervall zwischen dem Senden zweier Zeitreihen
  - Toleranz um die sich die Zeitreihe verspäten kann
  - Intervall zwischen zwei Messungen
  - Anzahl der Versuche im Pull-Betrieb bevor ein Alarm gesendet wird
  - Maximal erlaubte Abweichung vom Qualitätsmerkmal (Sensorvergleich)
  - Maximal erlaubte Abweichung vom Qualitätsmerkmal (Plausibilisierung)
  - Aktiv (ja/nein)

- Gewässer
  - Name
- Gewässerabschnitt
  - Name
- Geo-Referenz
  - Referenz-System
  - x
  - y
  - z
- WSA (Wasser- und Schifffahrtsamt)
  - Name
  - Beschreibung
- WSD (Wasser- und Schifffahrtsdirektion)
  - Name
  - Beschreibung
- Sensor
  - Sensornummer
  - Zeitstempel des letzten angekommenen Pushs
  - Zeitstempel des aktuellsten empfangenen Datenwertes
  - Aktiv (ja/nein)
- Parameter
  - Name
  - Parameter-Code
  - Einheit
- Rohdaten
  - Wert
  - Zeitstempel
  - Art der Daten (z.B. `contData`)
  - Qualitätsmerkmal (Sensorvergleich)
  - Plausibilisierung durchgeführt (ja/nein)
- Plausibilisierte Daten (Es werden nur Daten des plausibilisierten Sensors gespeichert)
  - Wert
  - Zeitstempel
  - Art der Daten (z.B. `contData`)
  - Qualitätsmerkmal (Sensorvergleich)

- Alarme
  - Alarm-Typ
  - Alarm-Status
  - Zeitstempel
  - Betroffener Sensor
  - Derzeitige Anzahl Benachrichtigungsversuche
  - Derzeitige Eskalationsstufe
  - Zeitstempel der letzten versendeten Benachrichtigung
- Benachrichtigungsvorlagen für Alarme
  - Alarm-Typ
  - Alarm-Status
  - Kontakt-Typ
  - Priorität
  - Betreff
  - Text
- Fehlende Daten
  - Letzter empfangener Datenwert vor diesem Eintrag
  - Start-Zeitstempel
  - End-Zeitstempel
  - Bisherige Anzahl Versuche im Pull-Betrieb
- Person
  - Vorname
  - Nachname
  - Anzahl der Benachrichtigungsversuche im Alarmfall
  - Intervall zwischen zwei Benachrichtigungen
- Kontaktadressen
  - Adresse
  - Kontakt-Typ
- Liste von Zuständigkeiten
  - Alarmtyp
  - Eskalationsstufe

4.2.2. Das konzeptuelle Datenmodell

Das konzeptuelle Datenmodell beschreibt den für die Datenmodellierung wichtigen Ausschnitt der realen Welt [KE06]. Die Beschreibung des konzeptuellen Modells geschieht mittels UML.

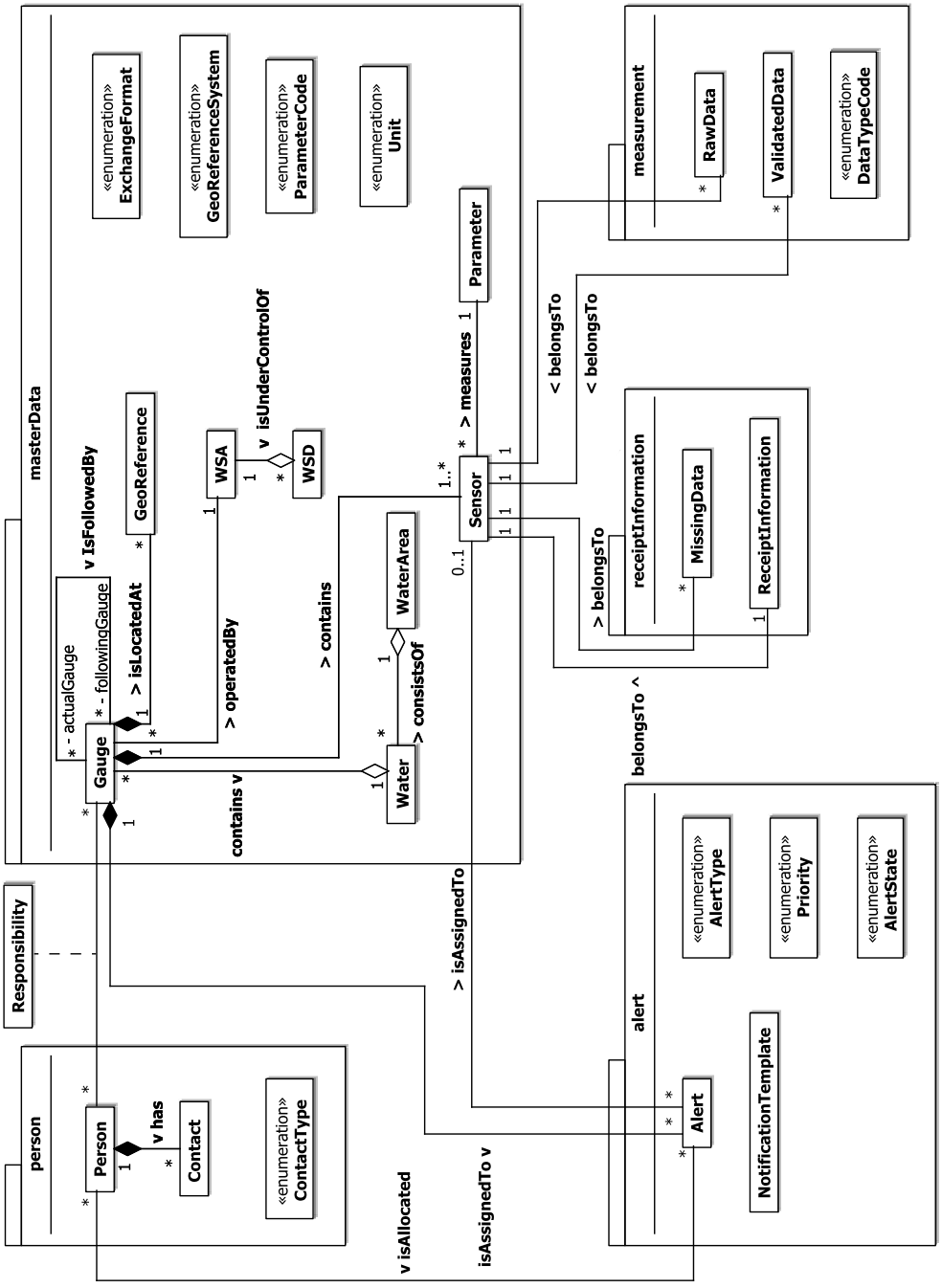


Abbildung 4.1.: Das konzeptuelle Datenmodell

Das konzeptuelle Datenmodell aus Abbildung 4.1 besteht aus den Paketen masterData, measurement, alert, person und receiptInformation. masterData enthält dabei alle Stammdaten eines Pegels, measurement beschreibt die Messdaten, alert enthält alle Informationen bezüglich ausgelöster Alarme, person beschreibt personenbezogene Daten und receiptInformation beschreibt zusätzliche Daten über den Empfang einer Zeitreihe.

Im Paket masterData sind die Klassen Gauge, Water, WaterArea, Sensor, Parameter, WSA, WSD und GeoReference, sowie die Enunmerations GeoReferenceSystem, ExchangeFormat, ParameterCode und Unit enthalten. measurement enthält die beiden Klassen RawData und ValidatedData sowie die Enumeration DataTypeCode. alert enthält die Klassen Alert und NotificationTemplate sowie die Enumerations AlertState, AlertType und Priority. Im Paket person sind die Klassen Person und Contact, sowie die Enumeration ContactType enthalten. receiptInformation enthält die Klassen MissingData und ReceiptInformation.

Die zentrale Klasse des Pakets masterData (siehe Abbildung 4.2) ist Gauge. Diese enthält die Attribute bezüglich Informationen zu den Pegeln (locationNumber, locationName, pnp und riverKilometer) und zu deren Konfiguration (ipAddress, getTimeSeriesPath, exchangeFormat equidistanceOfPushes, toleranceOfPushes, equidistanceOfMeasurements, maxPullRetries, maxSensorDeviation, maxValidationDeviation und isActive). locationNumber gibt dabei die Nummer des Standortes an, locationName dessen Namen, pnp den Pegelnullpunkt, riverKilometer die Stromkilometer, ipAddress die IP-Adresse des Pegels, getTimeSeriesPath den Pfad zum Web Service getTimeSeries, exchangeFormat das verwendete Austauschformat, equidistanceOfPushes das Intervall zwischen zwei Pushes, toleranceOfPushes die Zeit, um die sich ein Push verspäten kann, equidistanceOfMeasurements das Intervall zwischen zwei Messungen, maxPullRetries die maximale Anzahl an Versuchen im Pull-Betrieb, bevor ein Alarm gesendet wird, maxSensorDeviation die maximale Abweichung vom Qualitätsmerkmal beim Sensorvergleich, maxValidationDeviation die maximale Abweichung vom Qualitätsmerkmal bei der Plausibilisierung und isActive, ob der Pegel derzeitig aktiv ist. Für einige Attribute dieser Klasse müssen die Einheiten in denen sie angegeben sind, spezifiziert werden. Die Angabe von pnp, maxSensorDeviation und maxValidationDeviation geschieht in Zentimetern. equidistanceofPushes, toleranceOfPushes und equidistanceOfMeasurements wird in Minuten angegeben.

Jeder Pegel (Gauge) liegt genau in einem Gewässer (Water), wobei jedes Gewässer aus mehrere Pegeln besteht. Mehrere Gewässer werden wiederum zu

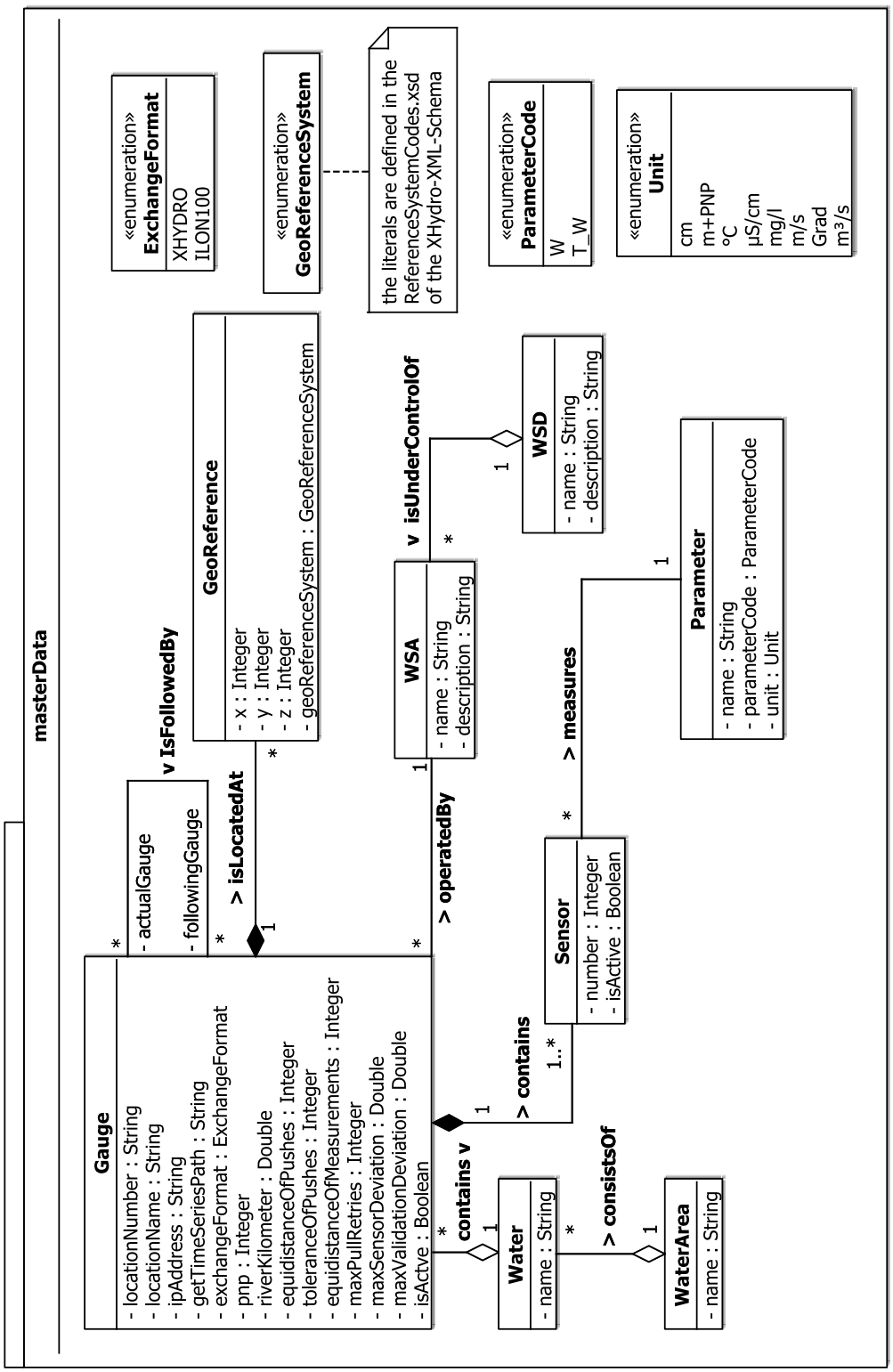


Abbildung 4.2.: Das konzeptuelle Datenmodell: `masterData`

einem Gewässerabschnitt (`WaterArea`) zusammengefasst, wobei auch hier ein Gewässer lediglich in einem Gewässerabschnitt liegen darf. `Water` und `WaterArea` enthalten dabei beide ein Attribut `name` für den Namen. Ein Pegel kann dabei keinen oder mehrere direkte Nachfolge-Pegel besitzen.

Der wichtigste Teil eines Pegels ist der Sensor (`Sensor`). Dieser enthält die Attribute `sensorNumber` für die Sensornummer und `isActive`, welches angibt, ob der Sensor derzeitig aktiv ist. Ein Pegel darf mehrere Sensoren haben, wobei jeder Sensor genau einen Parameter (`Parameter`) misst. Ein Parameter enthält Attribute für seinen Namen (`name`), den Parameter-Code (`ParameterCode`) und die Einheit der Messung (`unit`).

Weiterhin enthält ein Pegel keine oder mehrere Geo-Referenzen (`GeoReference`), welche den Standort weiter beschreiben können. Die Attribute `x`, `y` und `z` dienen zur Angabe von Punkten, mit denen sich 2D- oder 3D-Koordinaten darstellen lassen. Das Attribut `geoReferenceSystem` gibt das verwendete Geo-Referenz-System (Codeliste „ReferenceSystemCodes“; siehe [Bun09]) an.

Ein Pegel wird von einem Wasser- und Schifffahrtsamt (`WSA`) betrieben, welches einer Wasser- und Schifffahrtsdirektion (`WSD`) unterstellt ist. Eine WSD kontrolliert dabei mehrere WSAs, welche sich wiederum um mehrere Pegel kümmern können. Als Attribute enthalten diese Klassen einen Namen (`name`) sowie eine Beschreibung (`description`).

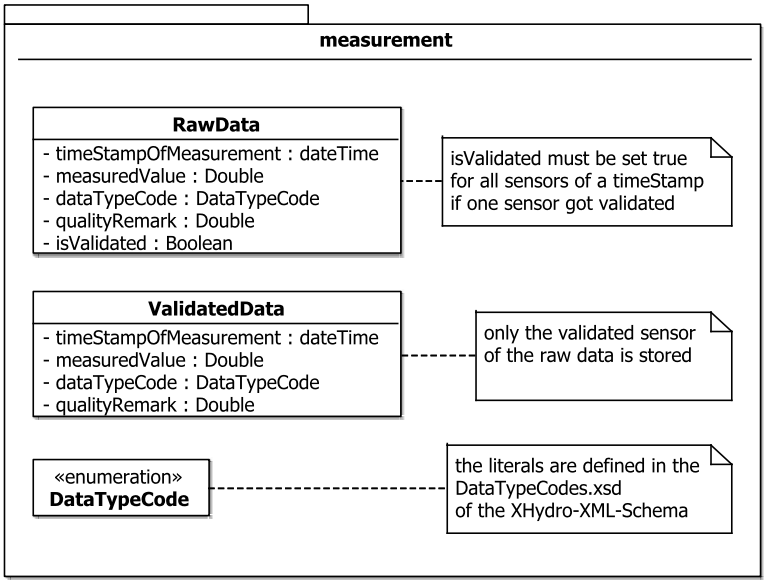


Abbildung 4.3.: Das konzeptuelle Datenmodell: `measurement`

Innerhalb des Pakets `measurement` (siehe Abbildung 4.3) werden die Messdaten in den Klassen `RawData` (Rohdaten) und `ValidatedData` (plausibilisierte

Daten) abgelegt. Beide enthalten als Attribute den eigentlichen Wert `measuredValue`, den zugehörigen Zeitstempel `timeStampOfMeasurement`, die Art der Daten (`dataTypeCode`) und das Qualitätsmerkmal des Sensorvergleichs `qualityRemark`. Die Rohdaten enthalten zusätzlich eine Information darüber, ob diese Daten bereits plausibilisiert wurden `isValidated`. Die Daten beider Klassen werden jeweils einem Sensor zugeordnet.

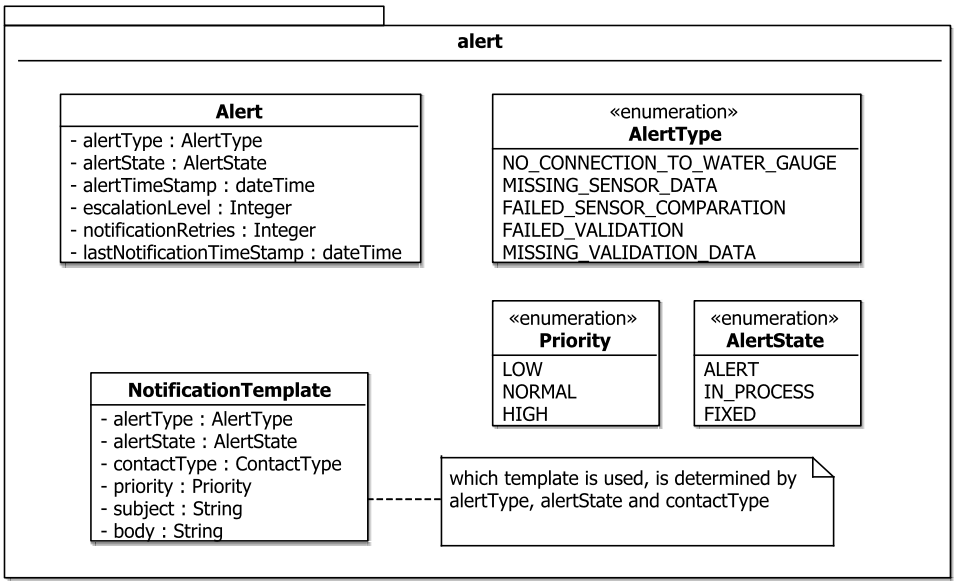


Abbildung 4.4.: Das konzeptuelle Datenmodell: `alert`

Ein Alarm wird durch die Klasse `Alert` im Paket `alert` (siehe Abbildung 4.4) beschrieben. Als Attribute enthält diese Klasse den Alarmtyp (`alertType`), den Alarm-Status (`alertState`), einen Zeitstempel (`alertTimeStamp`), die derzeitige Anzahl an Benachrichtigungsversuchen (`notificationRetries`), die derzeitige Eskalationsstufe (`escalationLevel`) und einen Zeitstempel, der angibt, wann der Alarm das letzten Mal gesendet wurde (`lastNotificationTimeStamp`). Jedem Alarm (`Alert`) werden keine oder mehrere Personen zugeordnet(`Person`; es sollte jedoch mindestens eine Person zugeordnet werden). Weiterhin wird einem Alarm der Pegel zugeordnet, welcher betroffen ist. Bei einigen Alarmfällen muss ebenfalls der betroffene Sensor angegeben werden. Ein Alarm kann maximal einen betroffenen Sensor haben, aber ein Sensor kann mehreren Alarmen zugeordnet sein. Die Klasse `NotificationTemplate` gibt die Informationen für die Benachrichtigungsvorlagen an. Als Attribute enthält diese den Alarmtyp (`alertType`), den Alarm-Status (`alertState`), den Kontakttyp (`contactType`), die Priorität (`priority`), einen Betreff (`subject`) und die eigentliche Nachricht (`body`).

Die Klasse `Person` im Paket `person` (siehe Abbildung 4.5) enthält die Attribu-

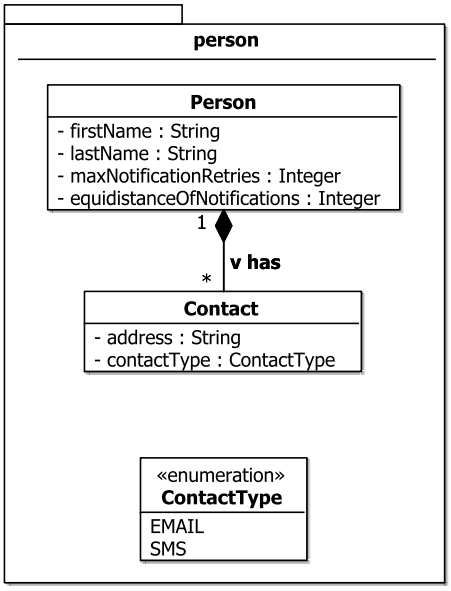


Abbildung 4.5.: Das konzeptuelle Datenmodell: `person`

te `firstName` für den Vornamen, `lastName` für den Nachnamen, `maxNotificationRetries` für die maximale Anzahl an Alarmbenachrichtigungsversuchen und `equidistanceOfNotifications` für den Abstand der Benachrichtigungen. Eine Person kann mehrere Kontaktadressen haben, welche in der Klasse `Contact` beschrieben werden. Eine Kontaktadresse enthält die Attribute `address` für die eigentliche Adresse, sowie `contactType` um den Typ der Adresse (E-Mail, SMS etc.) anzugeben. Eine Kontaktadresse gehört dabei zu genau einer Person. Weiterhin ist eine Person für bestimmte Pegel zuständig. Dies wird durch die Assozationsklasse `Responsibility` beschrieben. Als Attribute enthält diese den Alarmtyp (`alertType`) für den eine Person zuständig ist und die Eskalationsstufe (`escalationLevel`) in der sich die Person in diesem Fall befindet.

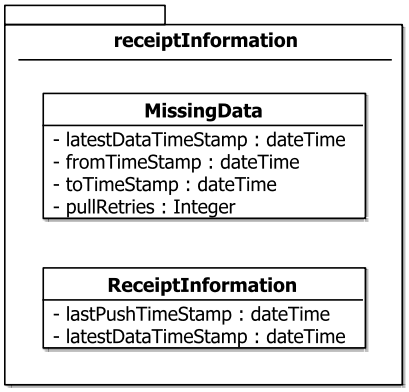


Abbildung 4.6.: Das konzeptuelle Datenmodell: `receiptInformation`

Im Paket `receiptInformation` (siehe Abbildung 4.6) enthält die Klasse `MissingData`, welche die fehlenden Daten einer Zeitreihe enthält. Sie besitzt die Attribute `latestDataTimeStamp`, welches den Zeitstempel des letzten Datenwertes vor dem Eintrag enthält, `fromTimeStamp` für den Start-Zeitstempel, für den die Daten fehlen, `toTimeStamp` den End-Zeitstempel und `pullRetries`, wodurch die derzeitige Anzahl der Versuche im Pull-Betrieb angegeben ist. Die fehlenden Daten gehören zu einem Sensor, wobei zu einem Sensor mehrere fehlende Daten gehören können. Ein Datum kann aber immer nur einem Sensor zugeschrieben werden. Die Klasse `ReceiptInformation` gibt Informationen über den letzten Datenerhalt an und enthält die Attribute `lastPushTimeStamp`, welche den Zeitstempel des letzten Pushs angibt und `latestDataTimeStamp`, welches den Zeitstempel des empfangene Datenwertes angibt. Ein Eintrag gehört dabei genau zu einem Sensor und ein Sensor besitzt auch genau einen Eintrag.

4.2.3. Überführung ins logische Datenmodell

Das logische Datenmodell beschreibt die Implementationsebene der Datenbank [KE06]. Im Gegensatz zum konzeptuellen Datenmodell lässt sich das logische Datenmodell direkt in die Datenbank überführen. Da eine relationale Datenbank verwendet werden soll, wird das konzeptuelle Datenmodell in diesem Abschnitt in ein relationales Datenmodell überführt.

Als erster Schritt werden die Klassen und Enumerations des konzeptuellen Modells übertragen. Dabei stellt jedes Attribut der Klasse eine Spalte der Tabelle dar. Für die Klassen, welche kein einen Dateneintrag identifizierendes Attribut enthalten, muss ein künstlicher Primärschlüssel im relationalen Modell eingefügt werden. Attribute, dessen Datentyp eine Enumeration ist, werden durch den Primärschlüssel der Enumeration ersetzt.

Im nächsten Schritt werden die Assoziationen und Assoziationsklassen des konzeptuellen Modells ins relationale Modell überführt. Dazu werden die Schlüssel aller beteiligten Klassen in eine Tabelle überführt. Bei Assoziationsklassen werden zusätzlich die Attribute der Assoziationsklasse übernommen.

Einige der auf diese Weise erstellten Tabellen lassen sich mit den Tabellen, welche aus den zugehörigen Klassen entstanden sind, zusammenführen. Beim Zusammenführen muss aber darauf geachtet werden, dass nur Tabellen mit gleichem Schlüssel zusammengefasst werden. Um herauszufinden, um welche Tabellen es sich handelt muss man auf die Multiplizitäten der Assoziation achten. Es werden die folgenden Assoziationstypen unterschieden (siehe Abbildung 4.7):

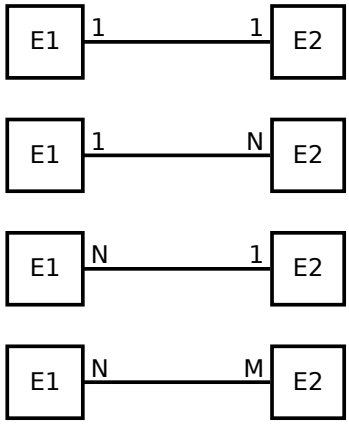


Abbildung 4.7.: Assoziationstypen im relationalen Datenmodell

1:1-Beziehung — Jeder Instanz der Klasse `E1` ist genau eine Instanz der Klasse `E2` zugeordnet. Ebenso ist jeder Instanz der Klasse `E2` genau eine Instanz der Klasse `E1` zugeordnet.

1:N-Beziehung (bzw. N:1-Beziehung) — Jeder Instanz der Klasse `E1` (`E2`) sind mehrere Instanzen der Klasse `E2` (`E1`) zugeordnet, jedoch ist jeder Instanz der Klasse `E2` (`E1`) genau eine Instanz der Klasse `E1` (`E2`) zugeordnet.

N:M-Beziehung — Jeder Instanz der Klasse `E1` sind mehrere Instanzen der Klasse `E2` zugeordnet. Ebenso sind jeder Instanz der Klasse `E2` mehrere Instanzen der Klasse `E1` zugeordnet.

Bei *1:1-Beziehungen* lässt sich die Tabelle der Assoziation eliminieren. Dies ist möglich, indem man den Schlüssel von `E2` in einer Spalte der Tabelle `E1` angibt. Ebenso könnte man aber auch den Schlüssel von `E1` als Spalte in der Tabelle `E2` angeben.

1:N-Beziehungen (bzw. *N:1-Beziehungen*) lassen sich ebenfalls eliminieren. Jedoch ist hier nicht so viel Freiraum geboten wie bei einer 1:1-Beziehung. Der Schlüssel der Tabelle `E2` (`E1`) muss dafür als Spalte in der Tabelle `E1` (`E2`) angegeben werden. Würde man stattdessen den Schlüssel der Tabelle `E1` (`E2`) in der Tabelle `E2` (`E1`) angeben, so wären mehrere Zeilen mit den gleichen Daten, jedoch unterschiedlichen Schlüsseln der eliminierten Tabelle vorhanden. Dadurch kann es zu Anomalien kommen.

N:M-Beziehungen lassen sich aufgrund von möglichen Anomalien definitiv nicht eliminieren.

Kann man bei den 1:1- und 1:N-Beziehungen Tabellen eliminieren, ist darauf zu achten, dass dadurch keine häufig auftretenden `NULL`-Werte entstehen. Dies ist jedoch bei dem konzeptuellen Modell aus Abschnitt 4.2.2 nicht der Fall.

Um weitere Anomalien zu verhindern müssen die Relationen noch in die *3. Normalform (3NF)* gebracht werden. Da dies aber bereits für das logische Modell, welches aus dem konzeptionellen Modell (siehe Abschnitt 4.2.2) erzeugt wurde, bis auf zwei gewollte Ausnahmen, der Fall ist, wird dies hier nicht weiter behandelt (siehe [KE06]).

[KE06]

Wendet man die oben beschriebenen Schritte auf das konzeptuelle Datenmodell in Abschnitt 4.2.2 an, so erhält man das folgende relationale Modell (auf die Angabe des Datentyps wird verzichtet):

Gauge: {[gaugeID, locationNumber, locationName, ipAddress, getTimeSeriesPath, exchangeFormatID, waterID, pnp, riverKilometer, wsaID, equidistanceOfPushes, toleranceOfPushes, equidistanceOfMeasurements, maxPullRetries, maxSensorDeviation, maxValidationDeviation, isActive]}
Water: {[waterID, name, waterAreaID]}
WaterArea: {[waterAreaID, name]}
GaugeIsFollowedByGauge: {[gaugeID, followingGaugeID]}
GeoReference: {[geoReferenceID, x, y, z, geoReferenceSystemID, gaugeID]}
WSA: {[wsaID, name, description, wsdID]}
WSD: {[wsdID, name, description]}
Sensor: {[sensorID, gaugeID, sensorNumber, parameterID, isActive]}
Parameter: {[parameterID, name, ,parameterCodeID, unitID]}
ParameterCode: {[parameterCodeID, parameterCode, description]}
Unit: {[unitID, unit, description]}
Priority: {[priorityID, priority, description]}
AlertType: {[alertTypeID, alertType, description]}
ContactType: {[contactTypeID, contactType, description]}
DataTypeCode: {[dataTypeCodeID, dataTypeCode, description]}
ExchangeFormat: {[exchangeFormatID, exchangeFormat, description]}
AlerState: {[alertStateID, alertState, description]}
GeoReferenceSystem: {[geoReferenceSystemID, geoReferenceSystem, description]}
RawData: {[sensorID, timeStampOfMeasurement, measuredValue, dataTypeCodeID, qualityRemark, isValidated]}
ValidatedData: {[sensorID, timeStampOfMeasurement, measuredValue, dataTypeCodeID, qualityRemark]}

Alert: {[alertID, alertTypeID, alertStateID, gaugeID, affectedSensorID, alertTimeStamp, escalationLevel, notificationRetries, lastNotificationTimeStamp]}
NotificationTemplate: {[alertTypeID, alertStateID, contactTypeID, priorityID, subject, body]}
MissingData: {[sensorID, latestDataTimeStamp, fromTimeStamp, toTimeStamp, pullRetries]}
ReceiptInformation: {[sensorID, lastPushTimeStamp, latestDataTimeStamp]}
Person: {[personID, firstName, lastName, maxNotifictionRetries, equidistanceOfNotification]}
Contact: {[contactID, personID, address, contactTypeID]}
Responsibility: {[personID, gaugeID, alertTypeID, escalationLevel]}

In der Tabelle *Gauge* dient das künstliche Attribut gaugeID als Schlüssel. Das Attribut exchangeFormat wurde durch die exchangeFormatID der Enumeration ersetzt. Neu hinzugekommen, sind die Attribute waterID und wsaID, welche aus der Eliminierung der Assoziationen deren Gauge——Water bzw. Gauge——WSA entstanden sind.

Die Tabelle *Water* besitzt den künstlich erzeugten Schlüssel waterID. Das Attribut waterAreaID ist eine Referenz auf die Tabelle *WaterArea*, welches anstatt einer eigenen Tabelle für die Assoziation Water——WaterArea verwendet wird.

Der Tabelle *WaterArea* wurde lediglich der künstliche Schlüssel waterAreaID hinzugefügt.

GaugeIsFollowedByGauge ist eine Tabelle, welche aus der N:M-Assoziation Gauge——Gauge entstanden ist. Als Schlüssel dient die Kombination aller Elemente der Tabelle (gaugeID und followingLocationNumber).

Eine Geo-Referenz aus der Tabelle *GeoReference* wird durch den künstlich erzeugten Schlüssel geoReferenceID bestimmt. Das Attribut locationNumber gibt den Pegel der Assoziation Gauge——GeoReference an. Die Assoziation GeoReference——GeoReferenceSystem wird durch das Attribut geoReferenceSystemID dargestellt.

Die Einträge in der Tabelle *WSA* werden durch den künstlichen Schlüssel wsaID identifiziert. Die Assoziation WSA——WSD lässt sich eliminieren. Zurück bleibt lediglich das Attribut wsdID.

Die Tabelle *WSD* erhält lediglich einen künstlichen Schlüssel (wsdID).

Die Tabelle *Sensor* erhält den künstlichen Schlüssel `sensorID`. Als Schlüssel wäre zwar ebenfalls die Kombination der Attribute `sensorNumber` und `gaugeID` in Frage gekommen, jedoch wurde der künstliche Schlüssel eingeführt, um beim Eliminieren von Assoziationen, welche `Sensor` betreffen, nicht immer beide Schlüsselattribute angeben zu müssen (siehe z.B. Tabelle *RawData* weiter unten). Die Attribute `gaugeID` und `parameterID` sind Referenzen, welche aus den Assoziationen `Gauge`——`Sensor` bzw. `Parameter`——`Sensor` entstanden sind.

In der Tabelle *Parameter* wird ein künstlicher Schlüssel zur Identifizierung der Einträge erstellt. Zudem werden die Attribute `parameterCode` und `unit` durch die IDs der Enumerations (`parameterCodeID` und `unitID`) ersetzt.

Die Enumerationen `ParameterCode`, `Unit`, `Priority`, `AlertType`, `ContactType`, `DataTypeCode`, `ExchangeFormat` und `AlertState` wurden alle als eigene Tabelle mit künstlichem Schlüssel erstellt. Weiterhin wird ein Attribut `description` eingeführt, welches die einzelnen Werte der Enumeration genauer beschreibt.

Für die Tabelle *RawData* werden die Attribute `sensorID` und `timeStampOfMeasurement` als Schlüssel verwendet. Das Attribut `dataTypeCode` wurde durch das Attribut `dataTypeCodeID` ersetzt, welches eine Referenz auf die zugehörige Enumeration *DataTypeCode* darstellt. Das Attribut `sensorID` ist aus der Assoziation `RawData`——`Sensor` entstanden. *RawData* ist nicht in 3NF. Schuld daran ist das Attribut `qualityRemark`, welches für alle Sensoren eines Pegels gleich ist. Da es jedoch nicht vorgesehen ist, dieses Attribut nach dem Speichern nochmals abzuändern, ist es nicht nötig diese Tabelle in 3NF anzugeben. Der Speicherverlust für die Referenz auf eine andere Tabelle wäre zu hoch.

Die Attribute `sensorID` und `timeStampOfMeasurement` identifizieren die Einträge der Tabelle *ValidatedData*. Bei dieser wurde ebenfalls das Attribut `dataTypeCode` durch das Attribut `dataTypeCodeID` ersetzt und die Assoziation `ValidatedData`——`Sensor` nach `sensorID` aufgelöst. Bezüglich der 3NF gilt das Selbe wie bei *RawData*.

Die Tabelle *Alert* erhält das künstliche Schlüsselattribut `alertID`, die Attribute `alertType` und `alertState` wurden durch die Referenzen `alertTypeID` und `alertState` auf die Enumerationen *AlertType* bzw. *AlertState* ersetzt und das Attribut `gaugeID` der Assoziation `Gauge`——`Alert` wurde hinzugefügt. Ebenso wurde das Attribut `affectedSensoID` aus der Assoziation `Sensor`——`Alert` hinzugefügt. Diese Tabelle enthält sowohl redundante Daten, als auch häufige `NULL`-Werte. Dies ist in diesem Fall beabsichtigt. Die benötigten Daten für einen Alarm sollen möglichst lokal an einem Ort (dieser Tabelle) zu finden

sein. Je nach Alarmtyp werden jedoch andere Attribute verwendet, was die NULL-Werte erklärt. Alarme werden nur solange gespeichert, wie sie bestehen. Danach werden die Daten aus der Tabelle gelöscht. Die redundanten Daten bzw. die Spalten mit NULL-Werten werden während dieser Zeit nicht verändert.

In der Tabelle *NotificationTemplate* sorgen die Attribute alertTypeID, alertStateID und contactTypeID für eine Identifizierung der Daten. Diese Attribute sind beim Ersetzen der Attribute alertTypeID, alertStateID und contactTypeID durch ihre Enumeration-Referenzen entstanden. Eine Ersetzung dieser drei Attribute durch einen künstlichen Schlüssel ist nicht möglich, da eine Vorlage nur anhand dieser drei Kriterien ausgewählt werden kann. Weiterhin wurde das Attribut priority durch die ID der Enumeration (priorityID) ersetzt.

Die Kombination der Attribute sensorID und latestDataTimeStamp dienen als Schlüssel der Tabelle *MissingData*. Das Attribut sensorID ist aus der Eliminierung der Assoziation Sensor——MissingData entstanden.

In der Tabelle ReceiptInformation dient das Attribut sensorID als Schlüssel, welches aus der Eliminierung der Assoziation Sensor——ReceiptInformation entstanden ist.

Die Tabelle *Person* wurde lediglich durch das künstliche Schlüsselattribut personID ergänzt.

Der Tabelle *Contact* wurde das künstliche Schlüsselattribut contactID hinzugefügt. Weiterhin wurde das Attribut contactType durch die Referenz contactTypeID der zugehörigen Enumeration *ContactType* ersetzt. Um die Assoziation Person——Contact zu eliminieren wurde das Attribut personID der Tabelle hinzugefügt.

Die Assoziationsklasse Responsibility wird durch die Tabelle *Responsibility* dargestellt. Als Schlüssel verwendet diese die Kombination der Attribute personID, gaugeID und alertTypeID.

Die N:M-Assoziation Person——Alert könnte man als eigene Tabelle *PersonIsAllocatedToAlert* hinzufügen. In diesem speziellen Fall lässt sich diese Assoziation aber auch über die Klasse Responsibility herstellen, da jedem Alarm genau ein Pegel zugewiesen wird.

4.2.4. Die SQL-Datei

Die Tabellen aus dem logischen Schema lassen sich direkt in SQL übersetzen. Das komplette SQL-Schema befindet sich in Abschnitt E.1.

```
1  CREATE TABLE Gauge
2  (
3    gaugeID INT NOT NULL AUTO_INCREMENT,
4    locationNumber TEXT NOT NULL,
5    locationName TEXT NOT NULL,
6    ipAddress TEXT NOT NULL,
7    getTimeSeriesPath TEXT NOT NULL,
8    exchangeFormatID INT NOT NULL,
9    waterID INT NOT NULL,
10   pnp DOUBLE,
11   riverKilometer DOUBLE,
12   wsaID INT NOT NULL,
13   equidistanceOfPushes INT NOT NULL,
14   toleranceOfPushes INT NOT NULL,
15   equidistanceOfMeasurements INT NOT NULL,
16   maxPullRetries INT NOT NULL,
17   maxSensorDeviation DOUBLE NOT NULL,
18   maxValidationDeviation DOUBLE NOT NULL,
19   isActive BOOLEAN NOT NULL,
20   PRIMARY KEY (gaugeID)
21 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
```

Listing 4.1: Tabelle `Gauge`

Listing 4.1 zeigt das SQL-Schema der Tabelle `Gauge`. Der Befehl `CREATE TABLE` in Zeile 1 erstellt die neue Tabelle. Die Zeilen 3 - 18 enthalten die einzelnen Spalten der Tabelle. Im Vergleich zum logischem Schema wurden die Befehle `NOT NULL` und `AUTO_INCREMENT` hinzugefügt. Durch die Angabe von `NOT NULL` (Zeilen 3 - 9 und 12 - 18) wird sichergestellt, dass diese Tabellen Spalte immer einen Wert enthalten muss. `AUTO_INCREMENT` (Zeile 3) dient zur Erzeugung einer eindeutigen Kennung der Tabellenzeile. Durch `PRIMARY KEY` Zeile 18 wird der Primarschlüssel der Tabelle angegeben. Mehrere Schlüssel werden dabei durch ein Komma getrennt. In Zeile 19 werden schließlich noch Datenbank spezifische Informationen angegeben. `ENGINE` gibt die Storage-Engine der Datenbank an. Für die PegelSuite-Datenbank wird gemäß den Anforderungen (siehe Anforderung **T-13** in Anhang D) InnoDB verwendet. InnoDB unterstützt im Gegensatz zu der Standard-Storage-Engine MyISAM Transaktionskontrolle sowie die referentielle Integrität der Fremdschlüssel [Ora10]. Der Zeichensatz der Tabelle wird durch `DEFAULT CHARSET` angegeben. Anforderung **T-14** schreibt hierfür sinnvollerweise UTF-8 vor. Das Schlüsselwort `COLLATION` gibt die Standard-Sortierfolge an [Ora10]. `utf8_unicode_ci` bedeutet, dass die Sortierung entsprechend [DW09] durchgeführt wird.

Die bereits angesprochene referentielle Integrität der Fremdschlüssel bzw. die Fremdschlüssel an sich wurden bisher noch nicht beachtet. Nachdem alle Tabellen erstellt wurden, ist es nötig diese zu definieren. Ein Beispiel ist in Listing 4.2 gezeigt.

```
255 ALTER TABLE Gauge ADD CONSTRAINT GaugeHasExchangeFormat
256   FOREIGN KEY (exchangeFormatID)
257    REFERENCES ExchangeFormat(exchangeFormatID)
258   ON DELETE RESTRICT ON UPDATE CASCADE
259 ;
```

Listing 4.2: Constraint `GaugeHasExchangeFormat`

Um die Informationen über den Fremdschlüssel der Tabelle hinzuzufügen, muss die Tabelle verändert werden. Dies geschieht durch den Befehl `ALTER TABLE` (Zeile 255). `ADD CONSTRAINT` (Zeile 255) fügt der Tabelle eine Bedingung hinzu. Durch `FOREIGN KEY` in Zeile 256 wird der Fremdschlüssel der Tabelle angegeben und `REFERENCES` in Zeile 257 gibt die Tabelle und den Primärschlüssel (in Klammern) an, auf welchen sich der Fremdschlüssel bezieht. Die beiden Befehle `ON DELETE` und `ON UPDATE` in Zeile 258 geben an, was mit den Daten der Tabelle geschieht, sobald die referenzierten Daten gelöscht bzw. verändert wurden. Im PegeSuite-SQL-Schema wurden die drei Varianten `CASCADE`, `RESTRICT` und `SET NULL` verwendet. `CASCADE` übernimmt die Änderungen auch für diese Tabelle, `RESTRICT` verbietet diese Änderungen und durch die Angabe von `SET NULL` wird der Fremdschlüssel bei einer Änderung auf `NULL` gesetzt.

4.3. Workflows

Dieses Kapitel beschriebt den Entwurf der einzelnen Workflows. Abschnitt 4.3.1 gibt einen Überblick über das Zusammenspiel der Workflows. In den folgenden Abschnitten werden die Implementationen der einzelnen BPEL-Prozesse sowie deren Schnittstellen zu den Java-Komponenten (siehe Abschnitt 4.4) beschrieben. Abschnitt 4.3.2 beschreibt die einzelnen Elemente der in den nachfolgenden Abschnitten verwendeten BPEL-Notation für die einzelnen Workflows. In Abschnitt 4.3.3 folgt die Beschreibung des 1. Workflows. Abschnitt 4.3.4 beschriebt den 2. Workflow, Abschnitt 4.3.5 den 3.Workflow und Abschnitt 4.3.6 den 4. Workflow. Die Schnittstellen zu den Java-Komponenten werden anschließend in Abschnitt 4.3.7 beschrieben.

4.3.1. Zusammenspiel der Workflows

In Abbildung 4.8 wird das Zusammenspiel der einzelnen Workflows dargestellt. Im Wartezustand (`wait`) können 4 Ereignisse eintreten, die jeweils einen unter-

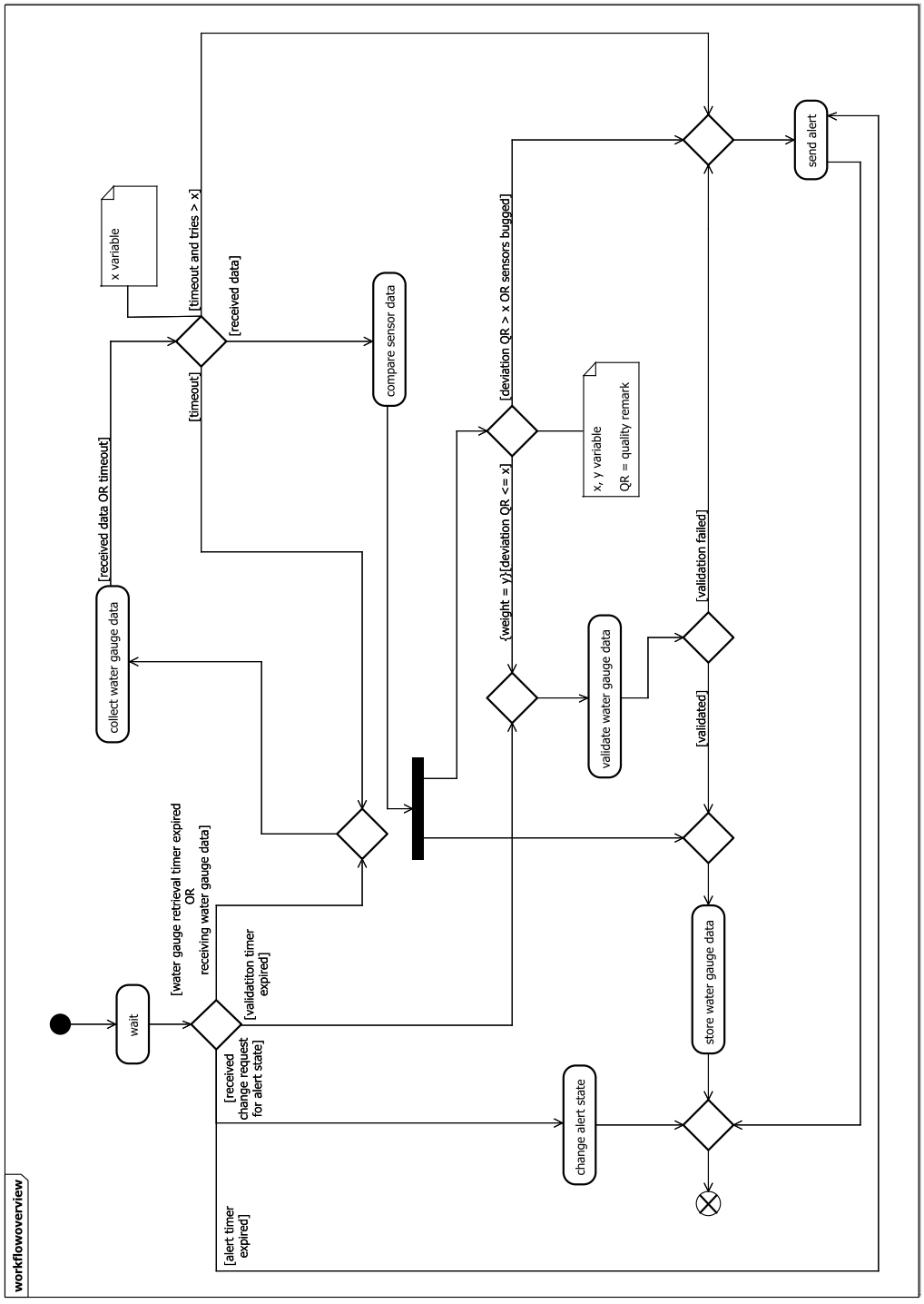


Abbildung 4.8.: Zusammenspiel der Workflows

schiedlichen Ablauf haben.

Wenn der Alarm-Timer abgelaufen ist (`[alert timer expired]`), dann wird lediglich ein weiterer Alarm `sendAlert` versendet.

Auch der Pfad `[received change request for alert state]` ist durch die Änderung des Alarmzustandes (`change alert state`) abgeschlossen.

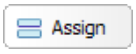
Ist der Plausibilisierungs-Timer (`[validation timer expired]`) abgelaufen, dann wird die Plausibilisierung durchgeführt (`validate water gauge data`). War die Plausibilisierung erfolgreich, dann wird die Zeitreihe als plausibilisiert in die Datenbank eingetragen (`store water gauge data`). Bei einem Fehlschlag der Plausibilisierung wird ein Alarm gesendet (send alert). Anschließend wird der Durchlauf beendet.

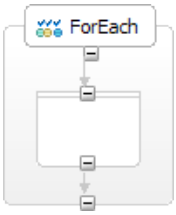
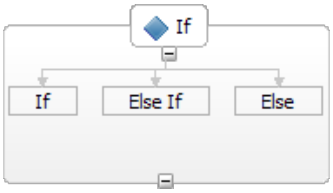
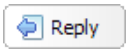
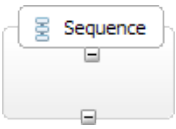
Wird die wichtigste Aktivität ausgeführt, das Empfangen oder Abrufen von Zeitreihen (`[water gauge retrieval timer expired OR receiving water gauge data]`), dann wird nach dem Empfang der Daten (`collect water gauge data`) das Qualitätsmerkmals anhand der einzelnen Sensoren berechnet (`compare sensor data`). Anschließend werden die Daten in der Datenbank gespeichert (`store water gauge data`) und das Qualitätsmerkmal überprüft. Ist die Abweichung des Qualitätsmerkmals zu groß oder einer der Sensoren fehlerhaft, dann wird ein Alarm gesendet (`send alert`). Ist die Abweichung im erlaubten Bereich und existieren genügend Daten um eine Plausibilisierung (`validate water gauge data`) zu starten, so wird diese wie bereits beschrieben durchgeführt.

Bei einer Abfrage der Pegeldaten kann es jedoch dazu kommen, dass der Pegel nicht erreichbar ist und daher keine Daten ankommen. Nach einem Timeout wird die Aktivität `collect water gauge data` erneut aufgerufen. Sollte dies nach `x` Versuchen nicht erfolgreich gewesen sein, wird ein Alarm versendet (`send alert`).

4.3.2. Die Verwendeten BPEL-Konstrukte

In den folgenden Abschnitten werden die Workflows durch Diagramme des BPEL-Designers von Eclipse dargestellt und anhand von diesen erläutert. Damit diese Diagramme lesbar sind, werden die einzelnen, verwendeten BPEL-Konstrukte kurz erläutert:

Element	Erklärung
	`assign` weist einer Variablen einen Wert zu. Dies kann durch das einfache Kopieren eines Wertes oder durch einen XPath 1.0 Ausdruck geschehen.

Element	Erklärung
	`empty` bewirkt nichts. Es dient in den Diagrammen in diesem Dokument lediglich als Kommentar.
	`forEach` führt Aktivitäten wiederholt aus.
	`if` dient der Fallunterscheidung. Um weitere Fälle hinzuzufügen existiert das Kindelement `elsif`. Alle restlichen unbehandelten Fälle können durch das Kindelement `else` behandelt werden.
	`invoke` ruft Partner Web Services auf. Dies können auch weitere BPEL-Prozesse sein.
	`receive` wartet auf das Eintreffen einer Nachricht.
	`reply` sendet eine Antwortnachricht auf ein vorangegangenes `receive`.
	`sequence` gruppiert mehrere Aktivitäten zu einer Sequenz. Die Aktivitäten werden hintereinander in der angegeben Reihenfolge ausgeführt.

4.3.3. Workflow 1: Datenabruf und -empfang

Der erste Workflow ist ein Zusammenspiel von den drei Prozessen `putTimeSeries`, `retrieveTimeSeries` und `processIncomingTimeSeries`.

`putTimeSeries` und `retrieveTimeSeries` besorgen die Daten im Push- bzw. Pull-Betrieb und leiten diese an den Prozess `processIncomingTimeSeries` weiter. `putTimeSeries` dient dabei als externe Schnittstelle, welche von den Pegeln verwendet wird, um die Daten pushen zu können. `retrieveTimeSeries` ist eine interne Schnittstelle, die von den zugehörigen Java-Komponenten von PegelSuite aufgerufen wird. `retrieveTimeSeries` ist dafür zuständig die Daten von den Pegeln abzurufen, wenn die Daten nicht als Push empfangen wurden. Der Prozess `retrieveTimeSeries` wird bei Ablauf eines Timers gestartet.

putTimeSeries

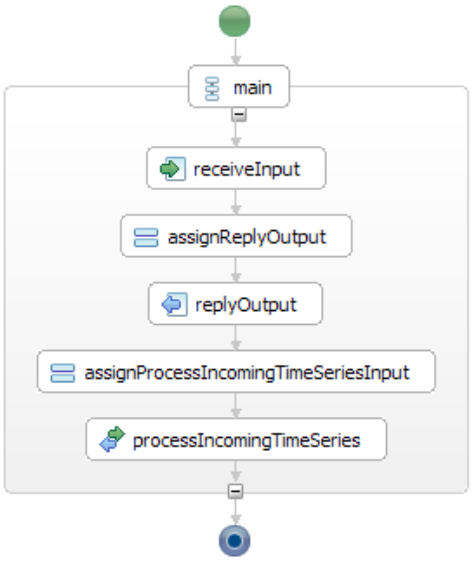


Abbildung 4.9.: Der Prozess `putTimeSeries`

Der Prozess `putTimeSeries` (siehe Abbildung 4.9) realisiert den Push-Betrieb. Eine eingehende Zeitreihe wird direkt an den Prozess `processIncomingTimeSeries` weitergeleitet.

Operation:	`putTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`tsel`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

`putTimeSeries` erwartet als Eingabe ein XHydro-Dokument (Wurzelelement `tsel`; siehe [Bun09]), sowie einen optionalen WS-Security-Header. Die Ausgabe besteht aus einem `string`, welcher als Empfangsbestätigung („`OK`“) genutzt wird, sowie ebenfalls aus einem WS-Security-Header. Der Eingabe-Parameter sowie der Ausgabe-Parameter werden normalerweise für die folgenden Schnittstellenspezifikationen von einem Elternelement *<Operationsname>* bzw. *<Operationsname>*`Response` umgeben. Dies gilt jedoch nicht für `putTimeSeries`, da diese Schnittstelle von der Bundesanstalt für Gewässerkunde bereits vorgegeben wurde.

retrieveTimeSeries

Der Prozess `retrieveTimeSeries` (siehe Abbildung 4.10) realisiert den Pull-Betrieb. Wird `retrieveTimeSeries` aufgerufen, so werden die Zeitreihen, der

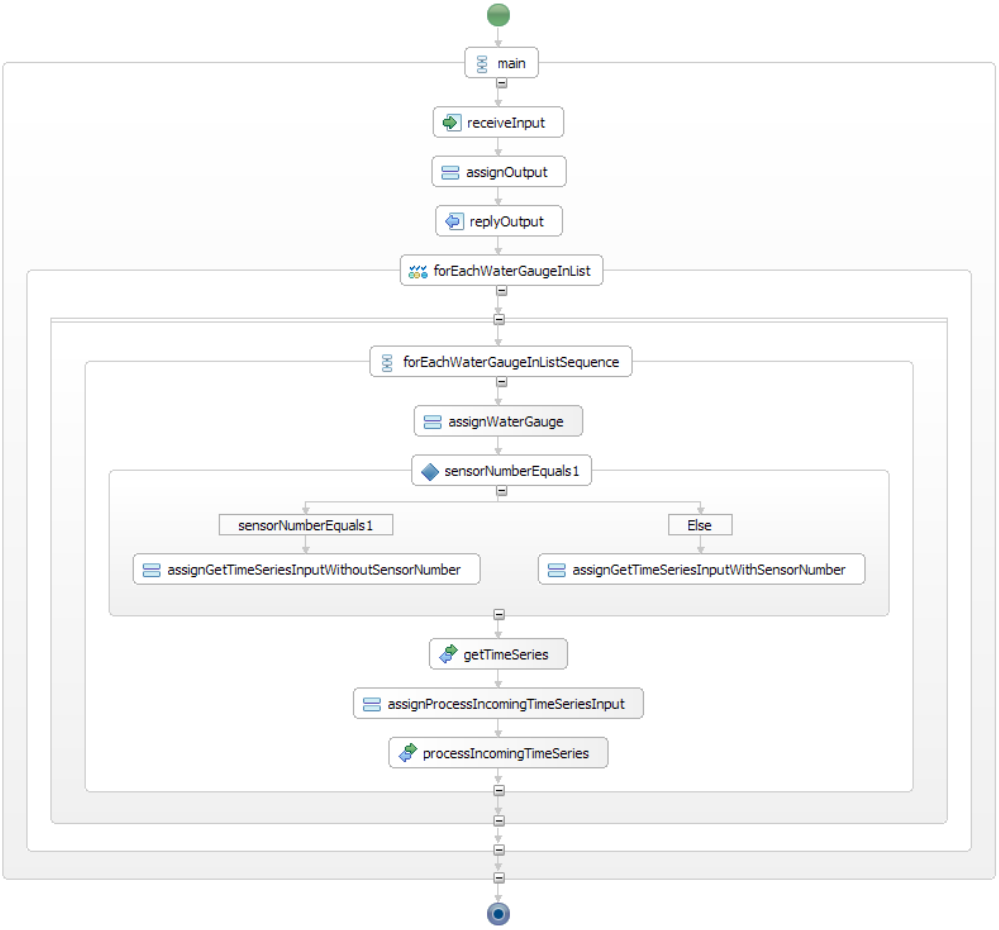


Abbildung 4.10.: Der Prozess `retrieveTimeSeries`

beim Aufruf übergebenen Pegel durch den Aufruf des Web Services `getTimeSeries` abgeholt. Dies muss für jeden Pegel, der beim Aufruf des Prozesses übergebenen Pegel-Liste (`waterGaugeList`; siehe Listing 4.3), durchgeführt werden (`forEachWaterGaugeInList`). Jede als Antwort eingehende Zeitreihe wird direkt an den Prozess `processIncomingTimeSeries` weitergeleitet. Bei der Anfrage an `getTimeSeries` ist allerdings zu beachten, dass für alle Sensoren die Form *<Parametercode><Sensornummer>* verwendet wird. Lediglich der erste Sensor wird durch *<Parametercode>* abgefragt (`sensorNumberEquals1`). Die erforderliche dynamische Änderung der IP-Adresse für den Aufruf von `getTimeSeries` ist derzeitig mit Apache ODE nicht möglich. Dies wurde durch einen Workaround in Java, welcher den eigentlichen Web Service aufruft gelöst.

Operation:	`retrieveTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGaugeList`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

`retrieveTimeSeries` erwartet als Eingabe-Parameter eine Liste von Pegeln (`waterGaugeList`; siehe Listing 4.3) und gibt einen `string` zurück, der als Empfangsbestätigung („`OK`“) genutzt wird. Als Ein- und Ausgabe-Header kann optional ein WS-Security-Header verwendet werden.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns:wg="http://watergauge.types.webservices.pegelsuite
6    .uni_koblenz.de"
7   xmlns="http://watergaugelist.types.webservices.pegelsuite
8    .uni_koblenz.de"
9   targetNamespace="http://watergaugelist.types.webservices
10   .pegelsuite.uni_koblenz.de">
11
12  <xsd:import namespace="http://watergauge.types.webservices
13   .pegelsuite.uni_koblenz.de"
14   schemaLocation="waterGauge.xsd" />
15
16  <xsd:element name="waterGaugeList" type="waterGaugeListType" />
17
18  <xsd:complexType name="waterGaugeListType">
19   <xsd:sequence>
20    <xsd:element ref="wg:waterGauge" maxOccurs="unbounded" />
21   </xsd:sequence>
22  </xsd:complexType>
```

```
23
24 </xsd:schema>
```

Listing 4.3: XML-Schema: `waterGaugeList`

In Listing 4.3 ist das XML-Schema von dem Element `waterGaugeList` aufgeführt. Das Element (Zeile 16) ist vom Datentyp `WaterGaugeListType` (Zeilen 18 - 22). Als Kindelement enthält `waterGaugeListType` eine Referenz auf das Element `waterGauge`. Dieses kann mehrmals vorkommen (Zeile 21).

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns="http://watergauge.types.webservices.pegelsuite
6    .uni_koblenz.de"
7   targetNamespace="http://watergauge.types.webservices
8    .pegelsuite.uni_koblenz.de">
9
10  <xsd:element name="waterGauge" type="waterGaugeType" />
11
12  <xsd:complexType name="waterGaugeType">
13   <xsd:sequence>
14    <xsd:element name="locationNumber" type="xsd:string" />
15    <xsd:element name="locationName" type="xsd:string" />
16    <xsd:element name="ipAddress" type="xsd:string" />
17    <xsd:element name="getTimeSeriesPath" type="xsd:string" />
18    <xsd:element name="pnp" type="xsd:double" />
19    <xsd:element name="sensorNumber" type="xsd:integer" />
20    <xsd:element name="sensorCount" type="xsd:integer" />
21    <xsd:element name="parameter" type="xsd:string" />
22    <xsd:element name="maxSensorDeviation" type="xsd:double" />
23    <xsd:element name="qualityRemark" type="xsd:double" />
24    <xsd:element name="from" type="xsd:dateTime" />
25    <xsd:element name="to" type="xsd:dateTime" />
26    <xsd:element name="timeStamp" type="xsd:dateTime" />
27   </xsd:sequence>
28  </xsd:complexType>
29
30 </xsd:schema>
```

Listing 4.4: XML-Schema: `waterGauge`

Das Element `waterGauge` ist in Listing 4.4 genauer beschrieben. Dieses Element (Zeile 10) ist vom Typ `waterGaugeType` (Zeile 12 - 28) und enthält die Kindelemente `locationNumber` (Standort-ID; Zeile 14), `locationName` (Standort-

Name; Zeile 15), `ipAddress` (IP-Adresse des Pegels; Zeile 16), `getTimeSeriesPath` (Pfad zu `getTimeSeries`; Zeile 17), `pnp` (Pegelnullpunkt; Zeile 18) `sensorNumber` (Sensornummer; Zeile 19), `sensorCount` (Sensoranzahl; Zeile 20), `parameter` (Parameter-Code ohne angehängte Sensornummer; Zeile 21), `maxSensorDeviation` (maximale Abweichung der Sensoren vom Qualitätsmerkmal; Zeile 22), `qualityRemark` (Qualitätsmerkmal; Zeile 23), `from` (Start-Zeitstempel; Zeile 24), `to` (End-Zeitstempel; Zeile 25) und `timeStamp` (Zeitstempel, falls es sich nicht um eine Zeitspanne handelt; Zeile 26).

Auch wenn alle Elemente laut Schema mindestens einmal vorkommen müssen, ist es nicht immer notwendig allen einen Wert zuzuweisen. Die Elemente müssen nur vorhanden sein, damit diese innerhalb eines BPEL-Prozesses hinzugefügt werden können, da das Erstellen eines Knotens mit BPEL nicht möglich ist. Dies gilt auch für alle weiteren in diesem Kapitel vorgestellten Schemas. Für den Prozess `retrieveTimeSeries` müssen die Elemente `locationNumber`, `ipAddress`, `getTimeSeriesPath`, `sensorNumber`, `parameter`, `from` und `to` allerdings zwingend angegeben werden.

processIncomingTimeSeries

Der Prozess `processIncomingTimeSeries` verarbeitet die von `putTimeSeries` bzw. `retrieveTimeSeries` weitergeleiteten Zeitreihen. Diese werden direkt gespeichert (`storeTimeSeries`), um Datenverlust vorzubeugen. Aus dieser Zeitreihe müssen nun für jeden Datenwert (`forEachDataValueInTimeSeries`) die `locationNumber`, welche einen Pegel eindeutig identifiziert, sowie der Zeitstempel der Daten extrahiert werden (`assignGetSensorDataListForOneElementInput`). Durch diese lassen sich die Daten aller Sensoren des Pegels für diesen Zeitpunkt abfragen (`getSensorDataList`). Anhand dieser Liste (siehe Listing 4.5) lässt sich die Anzahl der Sensoren, welche bereits Daten enthalten berechnen (`calculateSensorCountWithData`).

Nachdem nun alle benötigen Daten vorhanden sind, muss überprüft werden, ob die Anzahl der für diesen Zeitpunkt bereits erhaltenen Sensordaten (Element `sensorCountWithData`) mit der Anzahl der Sensoren des Pegels (Element `sensorCount`; Nachkomme von `sensorDataList`) übereinstimmt (`sensorCountEqualsSensorCountWithData`). Ist dies der Fall, so kann der Prozess `compareSensorData` (siehe Abschnitt 4.3.4) aufgerufen werden.

Operation:	`processIncomingTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`tsel`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

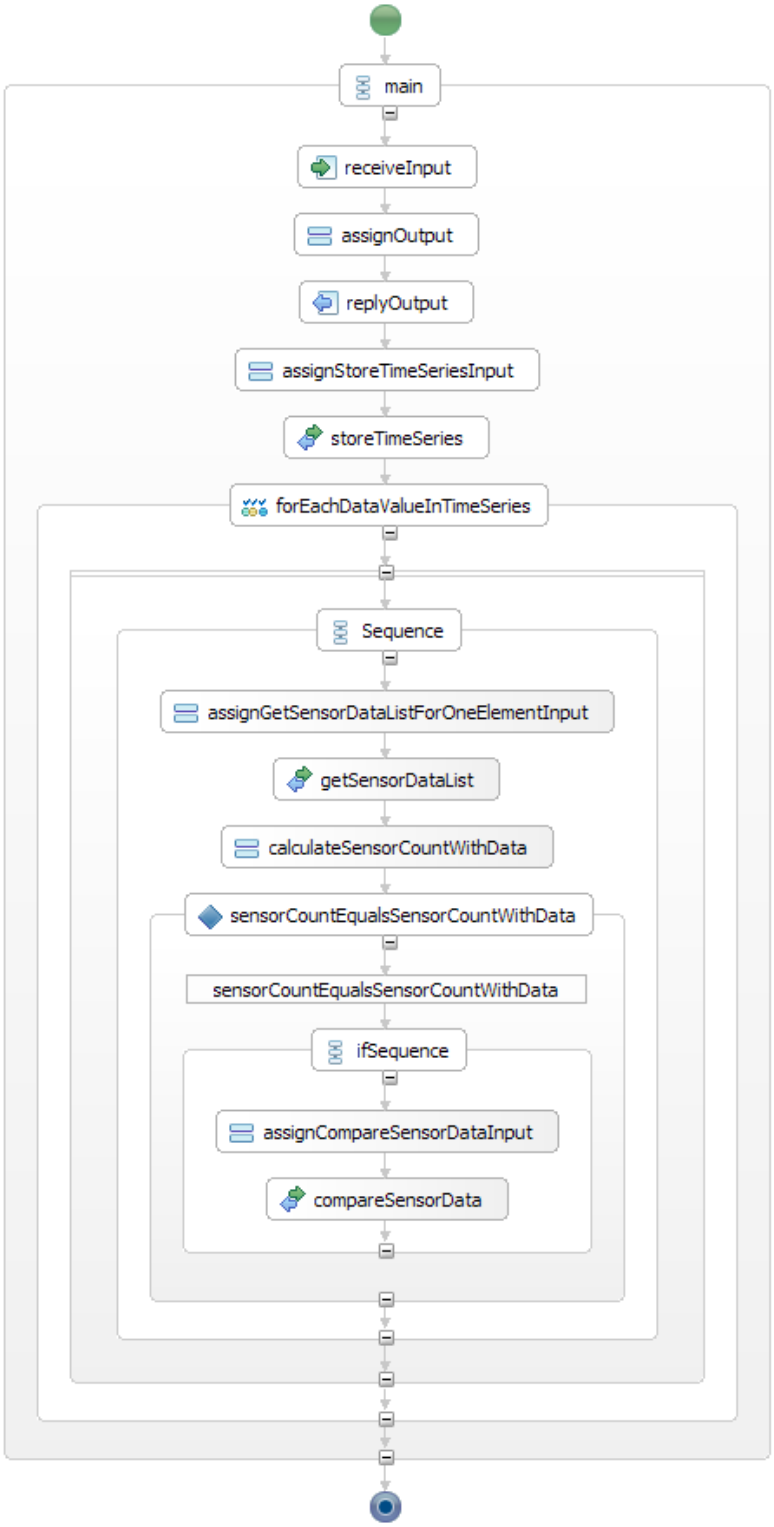


Abbildung 4.11.: Der Prozess `processIncomingTimeSeries`

`processIncomingTimeSeries` erwartet, wie auch schon `putTimeSeries`, als Eingabe ein XHydro-Dokument (Wurzelelement `tsel`; siehe [Bun09]), sowie einen optionalen WS-Security-Header. Die Ausgabe wird als Empfangsbestätigung („`OK`“) genutzt und kann ebenfalls einen WS-Security-Header enthalten.

4.3.4. Workflow 2: Sensorvergleich

Der zweite Workflow wird durch den Prozess `compareSensorData` realisiert. Dieser ist für das Vergleichen der Sensordaten zuständig und wird durch den Prozess `processIncomingTimeSeries` aufgerufen.

Nachdem Empfang der Sensordaten werden zunächst die im folgenden benötigten Variablen initialisiert. Anschließend wird in mehreren Schritten aus diesen das Qualitätsmerkmal berechnet. Dazu werden zuerst die Werte der einzelnen Parameter (`forEachSensorInTimeSeries`) addiert (`addValueToValueSum`) und die Anzahl der Werte ermittelt (`numberOfParameterWPlusPlus`). Im Anschluss wird daraus das Qualitätsmerkmal berechnet (`calculateQualityRemark`). Dieses wird dann in der Datenbank gespeichert (`storeQualityRemark`).

Die nächsten Aktivitäten berechnen die größte Abweichung der Sensordaten zum Qualitätsmerkmal. Dafür wird für jeden Sensor mit dem Parametercode W die Abweichung zum Qualitätsmerkmal berechnet (`calculateDeviationFromQualityRemark`). Die Berechnung kann nur erfolgen, falls der Wert in den vorher festgelegten Einheiten „cm“ oder „m+PNP“ angegeben ist. Da die Berechnung durch eine einfache Subtraktion geschieht, kann es sein, dass das Ergebnis im negativen Bereich liegt. In diesen Fall (`deviationFromQualityRemarkLess0`) wird der Betrag des Wertes berechnet (`calculateAbsoluteValueOfBiggestDeviationQualityRemark`). Wenn der berechnete Wert größer als die derzeitige größte Abweichung ist (`deviationFromQualityRemarkGreaterBiggestDeviationOfQualityRemark`), wird der alte Wert durch den Neuen ersetzt (`setNewBiggestDeviationFromQualityRemark`). Danach wird dieser Wert mit der maximal erlaubten Abweichung (`maxSensorDeviation`; Nachkomme von `sensorDataList`) verglichen. Sollte dabei die Abweichung der Sensordaten vom Qualitätsmerkmal größer sein, als die maximal erlaubte Abweichung (`biggestDeviationFromQualityRemarkGreaterMaxSensorDeviation`), muss ein neuer `FailedSensorComparation`-Alarm erstellt werden. Hierfür wird der Prozess `registerNewAlert` aufgerufen.

Operation:	`compareSensorData`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`sensorDataList`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

Der Prozess `compareSensorData` erwartet optional einen WS-Security-Header. Zudem ist die Angabe einer Liste der Sensordaten (`sensorDataList`) Pflicht. Die Ausgabe besteht aus einem optionalen WS-Security-Header und einem `string`, welcher als Empfangsbestätigung („`OK`“) dient.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns:wg="http://watergauge.types.webservices.pegelsuite
6    .uni_koblenz.de"
7   xmlns:sd="http://sensordata.types.webservices.pegelsuite
8    .uni_koblenz.de"
9   xmlns="http://sensordatalist.types.webservices.pegelsuite
10   .uni_koblenz.de"
11  targetNamespace="http://sensordatalist.types.webservices
12   .pegelsuite.uni_koblenz.de">
13
14  <xsd:import namespace="http://watergauge.types.webservices
15   .pegelsuite.uni_koblenz.de"
16   schemaLocation="waterGauge.xsd" />
17  <xsd:import namespace="http://sensordata.types.webservices
18   .pegelsuite.uni_koblenz.de"
19   schemaLocation="sensorData.xsd" />
20
21  <xsd:element name="sensorDataList" type="sensorDataListType" />
22
23  <xsd:complexType name="sensorDataListType">
24   <xsd:sequence>
25    <xsd:element ref="wg:waterGauge" />
26    <xsd:element ref="sd:sensorData" maxOccurs="unbounded" />
27   </xsd:sequence>
28  </xsd:complexType>
29
30  </xsd:schema>
```

Listing 4.5: XML-Schema: `sensorDataList`

Der Aufbau des Eingabe-Parameters `sensorDataList` wird in Listing 4.5 gezeigt. Das Element selbst (Zeile 21) ist vom Typ `sensorDataListType` (Zeilen 23 - 28). Als Kindelemente enthält es Referenzen auf die Elemente `waterGauge` (Zeile 25), welches bereits in Listing 4.4 beschrieben wurde, und `sensorData` (Zeile 26) (siehe Listing 4.6). Die obligatorischen Kindelemente von `waterGauge` sind `locationNumber`, `timeStamp`, `sensorCount` und `maxSensorDeviation`. Das Element `sensorData` darf beliebig oft, muss jedoch mindestens ein-

Abbildung 4.12.: Der Prozess `compareSensorData`

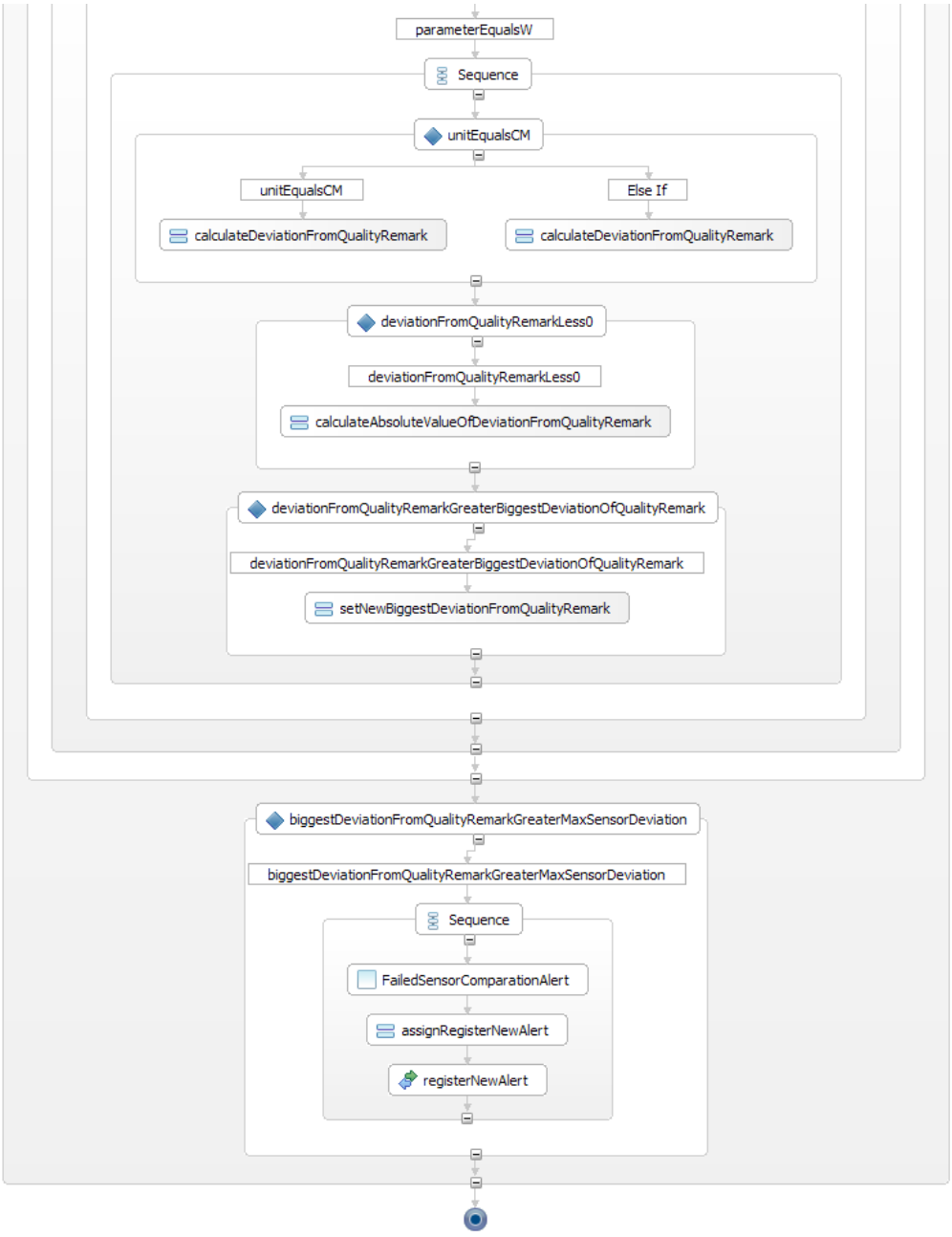


Abbildung 4.12: Der Prozess `compareSensorData` (Fortsetzung)

mal vorkommen. Jedes Vorkommen beschreibt die Daten zum zugehörigen Zeitstempel `timeStamp` (Kindelement von `waterGauge`) eines Sensors.

`sensorData` wird in Listing 4.6 dargestellt. Das Element (Zeile 10) ist vom Typ `sensorDataType` (Zeilen 12 - 19). Dieser Typ besitzt die Kindelemente `sensorNumber`, `parameter`, `value` und `unit` (Zeilen 14 - 17). `sensorNumber` gibt dabei die Nummer des Sensors und `parameter` dessen Parametercode ohne angehängte Sensornummer an. Das Element `value` gibt den Wert und `unit` dessen Einheit an.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns="http://sensordata.types.webservices.pegelsuite
6    .uni_koblenz.de"
7   targetNamespace="http://sensordata.types.webservices
8    .pegelsuite.uni_koblenz.de">
9
10  <xsd:element name="sensorData" type="sensorDataType" />
11
12  <xsd:complexType name="sensorDataType">
13   <xsd:sequence>
14    <xsd:element name="sensorNumber" type="xsd:integer" />
15    <xsd:element name="parameter" type="xsd:string" />
16    <xsd:element name="value" type="xsd:double" />
17    <xsd:element name="unit" type="xsd:string" />
18   </xsd:sequence>
19  </xsd:complexType>
20
21 </xsd:schema>
```

Listing 4.6: XML-Schema: `sensorData`

4.3.5. Workflow 3: Plausibilisierung

Der dritte Workflow wird durch den Prozess `validateWaterSector` realisiert. Dieser kümmert sich um die Plausibilisierung. Für jeden Pegel (`forEachWaterGaugeOfSector`) des Gewässerabschnitts wird nun die Plausibilisierung durchgeführt (`validateWaterGauge`). Ist der Wert plausibel (`isValidated`), so wird dieser den plausibilisierten Daten hinzugefügt (`storeValidatedTimeSeries`). Ist er es nicht, wird ein neuer `FailedValidation`-Alarm erzeugt (`registerNewAlert`).

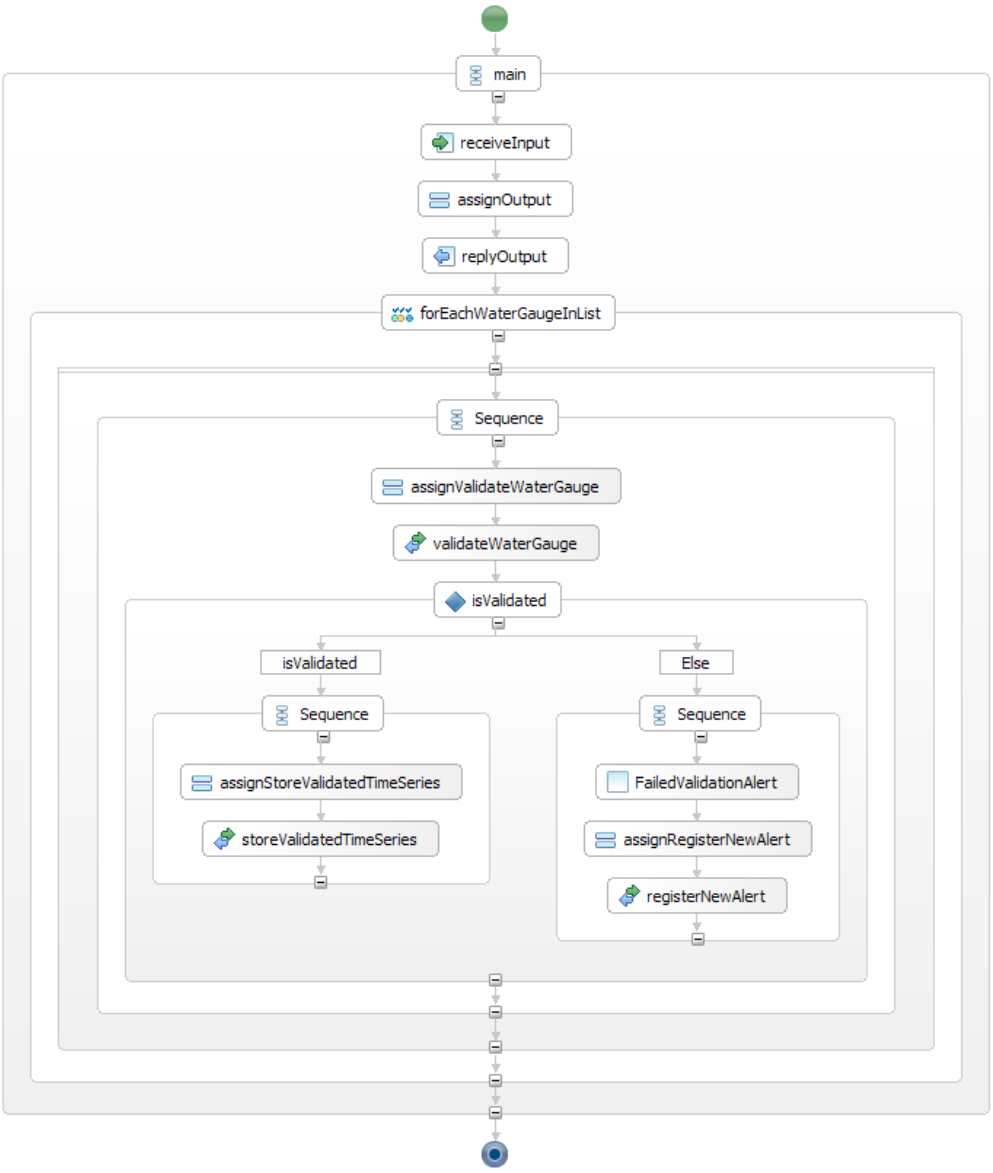


Abbildung 4.13.: Der Prozess `validateWaterSector`

Operation:	`validateWaterSector`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGaugeList`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

Wie auch alle bisherigen Prozesse erwartet `validateWaterSector` einen optionalen WS-Security-Header als Ein- bzw. Ausgabe-Header. Als obligatorischen Eingabe-Parameter erwartet der Prozess das Element `waterGaugeList`. Dieser Prozess benötigt nur die Elemente `locationNumber`, `sensorNumber` und `timeStamp`. Die Ausgabe des Prozesses wird als Empfangsbestätigung („`OK`“)

verwendet.

4.3.6. Workflow 4: Alarmmanagement

Der vierte Workflow wird durch die Prozesse `registerNewAlert` und `processAlert` realisiert. `registerNewAlert` registriert dabei einen neuen Alarm und ruft danach `processAlert` auf. In diesem Prozess wird der Alarm bearbeitet. `processAlert` kann nicht nur durch `registerNewAlert`, sondern auch durch einen Timer aufgerufen werden. Weiterhin existiert noch der Service `changeAlertState`, welcher den Alarmstatus auf Wunsch ändert.

`registerNewAlert`

`registerNewAlert` wird immer dann aufgerufen, wenn ein neuer Alarm auftritt. Nachdem dieser aufgerufen wurde, wird durch die Aktivität `checkForExistingAlert` überprüft, ob der Alarm bereits in der Datenbank gespeichert ist. Ist dies der Fall wird der Prozess beendet. Existiert der Alarm noch nicht, wird dieser in der Datenbank angelegt (`storeAlert`). Da für den Aufruf von `processAlert` noch ein paar Alarm-Daten fehlen (z.B. `maximalRetries`) wird der Alarm aktualisiert (`getAlert`). Mit diesen zusätzlichen Daten kann nun der Prozess `processAlert` aufgerufen werden.

Operation:	`registerNewAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

Der Prozess `registerNewAlert` erwartet als Eingabe ein `alert`-Element. Die Ausgabe besteht aus einem `string`, welcher als Empfangsbestätigung („`OK` “) dient. Optional kann ein WS-Security-Header der Ein- bzw. Ausgabe hinzugefügt werden.

Das Element `alert` (Zeile 9; Listing 4.7) enthält die Informationen zu den Alarmen. Es ist vom Typ `alertType` (Zeilen 11 - 24). Dieser Datentyp enthält die Kindelemente `alertType` (Zeile 13) , `locationNumber` (Zeile 14), `affectedSensor` (Zeile 15), `timeStamp` (Zeile 16), `currentRetries` (Zeile 17), `maximalRetries` (Zeile 18), `currentEscalationLevel` (Zeilen 19 und 20) und `maximalEscalationLevel` (Zeilen 21 und 22). `alertType` gibt den Alarm-Typ an, `locationNumber` die Standort-ID des Pegels, `affectedSensor` die Nummer des betroffenen Sensors, `timeStamp` den Zeitstempel der Alarmzeit,

Abbildung 4.14.: Der Prozess `registerNewAlert`

`currentRetries` die derzeitige Anzahl an Benachrichtigungsversuchen, `maximalRetries` die maximale Anzahl an Benachrichtigungsversuchen, `currentEscalationLevel` die derzeitige Eskalationsstufe und `maximalEscalationLevel` die maximale Eskalationsstufe.

```
1 <?xml version="1.0" encoding="utf-16"?>
2 <xsd:schema attributeFormDefault="unqualified"
3  elementFormDefault="qualified" version="1.0"
```

```
4  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5  xmlns="http://alert.types.webservices.pegelsuite.uni_koblenz.de"
6  targetNamespace=
7   "http://alert.types.webservices.pegelsuite.uni_koblenz.de">
8
9  <xsd:element name="alert" type="alertType" />
10
11 <xsd:complexType name="alertType">
12  <xsd:sequence>
13   <xsd:element name="alertType" type="xsd:string" />
14   <xsd:element name="locationNumber" type="xsd:string" />
15   <xsd:element name="affectedSensor" type="xsd:integer" />
16   <xsd:element name="timeStamp" type="xsd:dateTime" />
17   <xsd:element name="currentRetries" type="xsd:integer" />
18   <xsd:element name="maximalRetries" type="xsd:integer" />
19   <xsd:element name="currentEscalationLevel"
20    type="xsd:integer" />
21   <xsd:element name="maximalEscalationLevel"
22    type="xsd:integer" />
23  </xsd:sequence>
24 </xsd:complexType>
25
26 </xsd:schema>
```

Listing 4.7: XML-Schema: `alert`

Je nach Alarmtyp, werden andere Informationen benötigt:

NoConnectionToWaterGauge — alertType, locationNumber, timeStamp, currentRetries, maximalRetries, currentEscalationLevel, maximalEscalationLevel

MissingSensorData — alertType, locationNumber, affectedSensor, timeStamp, currentRetries, maximalRetries, currentEscalationLevel, maximalEscalationLevel

FailedSensorComparation — alertType, locationNumber, timeStamp, currentRetries, maximalRetries, currentEscalationLevel, maximalEscalationLevel

FailedValidation — alertType, locationNumber, affectedSensor, timeStamp, currentRetries, maximalRetries, currentEscalationLevel, maximalEscalationLevel

MissingValidationData — alertType, locationNumber, affectedSensor, timeStamp, currentRetries, maximalRetries, currentEscalationLevel, maximalEscalationLevel

Die unterstrichenen Daten sind nötig, um einen Alarm eindeutig zu identifizieren. Dies wird z.B. für `checkForExistingAlert` benötigt.

Für den Prozess `registerNewAlert` fallen bei allen Alarmtypen die folgenden Elemente weg: `currentRetries`, `maximalRetries`, `currentEscalationLevel`, `maximalEscalationLevel`.

processAlert

Die Verarbeitung von Alarmen geschieht durch den Prozess `processAlert`. Nach dem Aufruf von `processAlert` wird überprüft, ob die maximal erlaubte Anzahl an Wiederholungen (Element `maximalRetries`) kleiner der derzeitigen Wiederholungsanzahl (Element `currentRetries`) ist (`currentRetriesLessMaximalRetries`). Ist dies der Fall, dann wird die derzeitige Wiederholungsanzahl erhöht (`valuePlusPlus`). Sollte dies nicht der Fall sein, so muss die Eskalationsstufe gewechselt werden. Auch hier muss die derzeitige Stufe (`currentEscalationLevel`) und die maximal erlaubte Stufe (`maximalEscalationLevel`) verglichen werden (`currentEscalationLevelLessMaximalEscalationLevel`). Ist der derzeitige Wert niedriger als der maximal erlaubte, so muss der Eskalationslevel erhöht werden (`valuePlusPlus`) und die Anzahl der Versuche muss wieder auf 0 gesetzt werden (`setCurrentRetriesTo0`). Die Aktivität `valuePlusPlus` ist nötig, da die verwendete Version von Apache ODE durch einen Bug, bei einer Berechnung von `integer`-Werten mit XPath, ohne Berücksichtigung des Variablentyps, einen `double`-Wert zurück gibt. Daher muss diese Funktion in Java durchgeführt werden.

In beiden Fällen müssen die veränderten Alarmdaten in der Datenbank aktualisiert werden (`updateAlert`). Anschließend werden die Benachrichtigungsdaten angefragt (`getNotificationData`). Mit diesen Informationen kann ein Alarm versendet werden (`sendAlert`).

Operation:	`processAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`string`

`processAlert` erwartet als optionalen Eingabe-Header einen WS-Security-Header. Dies ist ebenfalls beim Ausgabe-Header der Fall. Der Eingabe-Parameter selbst

Abbildung 4.15.: Der Prozess `processAlert`

ist das Element `alert`. Für diesen Prozess werden Elemente des jeweiligen Alarmtyps benötigt. Die Ausgabe besteht aus einem `string`, welcher als Empfangsbestätigung („`OK`“) dient.

changeAlertState

Der Web Service `changeAlertState` dient dazu den Alarmzustand zu ändern. Dies geschieht durch einen REST-Aufruf über die HTTP-GET-Methode durch einen Link in einer Alarm-E-Mail.

Operation:	`processAlert`
Eingabe-Parameter:	`alertID`, `newAlertStateID`
Ausgabe-Parameter:	HTML-Dokument

Als Eingabe benötigt `changeAlertState` die beiden Parameter `alertID` und `newAlertStateID` vom Typ `integer`. Bei der Ausgabe handelt es sich um ein HTML-Dokument, welches die Änderung bestätigt.

4.3.7. Schnittstellen zu aufgerufenen Web Services

Bei der Abarbeitung der Geschäftsprozesse kommt es immer wieder dazu, dass BPEL-externe Web Services (in Java implementiert) aufgerufen werden. Dies ist beispielsweise beim Speichern und Abrufen von Daten in/aus einer Datenbank der Fall. Die Schnittstellen dieser Web Services werden im Folgenden spezifiziert.

getTimeSeries

Operation:	`getTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`getTimeSeriesRequest`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`tsel`

Durch `getTimeSeries` können Zeitreihen bei dem jeweiligen Pegel abgerufen werden. Als Eingabe erwartet `getTimeSeries` optional einen WS-Security-Header, sowie einen getTimeSeriesRequest (siehe [Bun09]). Die Ausgabe besteht aus einem optionalen WS-Security-Header und einem XHydro-Dokument (Wurzelelement `tsel`; siehe [Bun09]). Die Elternelemente *`Operationsname`* bzw. *`Operationsname`*`Response` entfallen bei diesem Web Service.

Da Apache ODE derzeitig keine dynamischen Invokes, d.h. Invokes mit dynamisch zugewiesener IP-Adresse, unterstützt. Musste für diesen Web Service ein

Workaround in Java implementiert werden, der anschließend den eigentlichen Web Service aufruft.

storeTimeSeries

Operation:	`storeTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`tsel`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Der Web Service `storeTimeSeriesList` ist für das Speichern der Zeitreihe in der Datenbank zuständig. Als Eingabe erwartet `storeTimeSeriesList` optional einen WS-Security-Header, sowie ein XHydro-Dokument (Wurzelelement `tsel`; siehe [Bun09]). Die Ausgabe besteht aus dem optionalen WS-Security-Header.

getSensorDataList

Operation:	`getSensorDataList`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGauge`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`sensorDataList`

Der Service `getSensorDataList` gibt eine Liste von Rohdaten eines jeden Sensors des angefragten Pegels zu dem angegebenen Zeitstempel aus der Datenbank zurück. Als Eingabe- und Ausgabe-Header kann ein WS-Security-Header verwendet werden. Die Eingabe erwartet ein `waterGauge`-Element (siehe Listing 4.4). Benötigt werden die Kindelemente `locationNumber` und `timeStamp` von `waterGauge`. Die Ausgabe enthält ein `sensorDataList`-Element (siehe Listing 4.5).

storeQualityRemark

Operation:	`storeQualityRemark`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGauge`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Durch `storeQualityRemark` kann das Qualitätsmerkmal des Sensorvergleichs in der Datenbank gespeichert werden. Die Eingabe des Web Services `storeQualityRemark` erwartet den optionalen WS-Security-Header, sowie das Element `waterGauge` (siehe Listing 4.4). Benötigt werden dessen Kindelemente `locationNumber`, `timeStamp` und `qualityRemark`. Die Ausgabe besteht aus einem optionalem WS-Security-Header.

validateWaterGauge

Operation:	`validateWaterGauge`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGauge`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`boolean`

Der Web Service `validateWaterGauge` ist dafür zuständig, die Plausibilisierung aufzurufen. Dieser Web Service gehört zum Java-Teil von PegelSuite und bereitet den Aufruf der Plausibilisierung vor, indem er alle benötigten Daten sammelt und der Plausibilisierung übergibt. Damit dieser weiß, welcher Pegel plausibilisiert werden soll, benötigt er aber bereits als Eingabe das Element `waterGauge` (siehe Listing 4.4), genauer gesagt, dessen Kindelemente `locationNumber`, `sensorNumber` und `timeStamp`. Die Ausgabe des Services ist ein Wahrheitswert, welcher angibt, ob die Daten erfolgreich plausibilisiert werden konnten. Als optionaler Header kann sowohl für die Eingabe, als auch für die Ausgabe ein WS-Security-Header verwendet werden.

storeValidatedTimeSeries

Operation:	`storeValidatedTimeSeries`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`waterGauge`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Durch `storeValidatedTimeSeries` werden die plausibilisierten Daten in der Datenbanktabelle für plausibilisierte Daten gespeichert. Die Eingabe des Web Services `storeValidatedData` erwartet den optionalen WS-Security-Header, sowie das Element `waterGauge` (siehe Listing 4.4). Die Angabe der Elemente `locationNumber`, `sensorNumber` und `timeStamp` ist hierfür ebenfalls notwendig. Die Ausgabe besteht aus einem optionalem WS-Security-Header.

checkForExistingAlert

Operation:	`checkForExistingAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`boolean`

`checkForExistingAlert` überprüft, ob der Alarm bereits in der Datenbank vorhanden ist. Er erwartet als Ein- und Ausgabe-Header einen optionalen WS-Security-Header. Weiterhin besteht die Eingabe aus dem Element `alert`. Verwendet werden die Elemente des jeweiligen Alarmtyps mit Ausnahme von `currentRetries`, `maximalRetries`, `currentEscalationLevel` und `maximalEscalationLevel`. Die Ausgabe besteht aus einem Wahrheitswert, welcher angibt, ob der Alarm bereits existiert.

storeAlert

Operation:	`storeAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Der Web Service `storeAlert` speichert einen neuen Alarm in der Datenbank. Dafür erhält er als Eingabe ein `alert`-Element (siehe Listing 4.7). Benötigt werden die Daten des jeweiligen Alarmtyps mit Ausnahme von `currentRetries`, `maximalRetries`, `currentEscalationLevel` und `maximalEscalationLevel`. Weiterhin kann ein optionaler WS-Security-Header hinzugefügt werden. Dieser kann auch in der Ausgabe verendet werden.

getAlert

Operation:	`getAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`alert`

`getAlert` gibt eine aus der Datenbank aktualisierte Version des in der Eingabe angegebenen Alarms zurück. Dies ist nötig um den Alarm mit bisher unbekannte Elemente wie `currentRetries` zu erweitern. Als Eingabe sowie als Ausgabe muss ein `alert`-Element angegeben werden. Zusätzlich kann auch hier der WS-Security-Header angegeben werden.

valuePlusPlus

Operation:	`valuePlusPlus`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`integer`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`integer`

Wie bereits erwähnt dient der Web Service `valuePlusPlus` als Workaround für einen Bug in Apache ODE, welcher bei einer Berechnung mittels XPath einen `double`-Wert statt des erwarteten `integer`-Werts liefert. Ein- und Ausgabe bestehen aus einem `integer`-Wert sowie optional aus einem WS-Security-Header.

updateAlert

Operation:	`updateAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Der Web Service `updateAlert` aktualisiert die Alarmdaten in der Datenbank. Als Eingabe-Header kann optional ein WS-Security-Header verwendet werden. Weiterhin besteht die Eingabe aus dem Element `alert` (siehe Listing 4.7). Benötigt werden alle Kindelemente des jeweiligen Alarmtyps. Die Ausgabe besteht aus einem optionalen WS-Security-Header.

getNotificationList

Operation:	`getNotificationList`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`alert`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`notificationList`

Der Web Service `getNotificationData` gibt alle Kontaktadressen für einen Alarm aus der Datenbank zurück. Hierfür muss die Eingabe aus dem Element `alert` (siehe Listing 4.7) bestehen. Benötigt werden alle Elemente des jeweiligen Alarmtyps, mit Ausnahme von `maximalRetries` und `maximalEscalationLevel`. Die Ausgabe besteht aus dem Element `notificationList`. Optional kann der Eingabe bzw. Ausgabe ein WS-Security-Header hinzugefügt werden.

```
1 <?xml version="1.0" encoding="utf-16"?>
2 <xsd:schema attributeFormDefault="unqualified"
3  elementFormDefault="qualified" version="1.0"
```

```
4  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5  xmlns:n="http://notification.types.webservices.pegelsuite
6   .uni_koblenz.de"
7  xmlns="http://notificationlist.types.webservices.pegelsuite
8   .uni_koblenz.de"
9  targetNamespace="http://notificationlist.types.webservices
10  .pegelsuite.uni_koblenz.de">
11
12 <xsd:import schemaLocation="notification.xsd" namespace="http://
13  notification.types.webservices.pegelsuite.uni_koblenz.de" />
14
15 <xsd:element name="notificationList"
16  type="notificationListType" />
17
18 <xsd:complexType name="notificationListType">
19  <xsd:sequence>
20   <xsd:element ref="n:notification" maxOccurs="unbounded" />
21  </xsd:sequence>
22 </xsd:complexType>
23
24 </xsd:schema>
```

Listing 4.8: XML-Schema: `notificationList`

Das Element `notificationList` (Zeilen 15 und 16 in Listing 4.8) vom Typ `notificationListType` (Zeilen 18 - 22) enthält eine Referenz auf das Kindelement `notification` (Zeile 20). Dieses Element kann beliebig oft, muss jedoch mindestens einmal, vorkommen.

```
1  <?xml version="1.0" encoding="utf-16"?>
2  <xsd:schema attributeFormDefault="unqualified"
3   elementFormDefault="qualified" version="1.0"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns="http://notification.types.webservices.pegelsuite
6    .uni_koblenz.de"
7   targetNamespace="http://notification.types.webservices
8    .pegelsuite.uni_koblenz.de">
9
10  <xsd:element name="notification" type="notificationType" />
11
12  <xsd:complexType name="notificationType">
13   <xsd:sequence>
14    <xsd:element name="address" type="xsd:string" />
15    <xsd:element name="contactType" type="xsd:string" />
16    <xsd:element name="priority" type="xsd:string" />
17    <xsd:element name="subject" type="xsd:string" />
```

```
18      <xsd:element name="body" type="xsd:string" />
19     </xsd:sequence>
20    </xsd:complexType>
21
22  </xsd:schema>
```

Listing 4.9: XML-Schema: `notification`

`notification` (Zeile 10 in Listing 4.9) ist vom Typ `notificationType` (Zeilen 12 - 20) und besitzt mehrere Kindelemente: `address` (Zeile 14), `contactType` (Zeile 15), `priority` (Zeile 16), `subject` (Zeile 17) und `body` (Zeile 18). Die Adresse wird dabei durch das Element `address`, dessen Typ durch `contactType`, die Priorität durch das Element `priority`, der Betreff durch das Element `subject` und der eigentliche Benachrichtigungstext durch das Element `body` angegeben.

Der Benachrichtigungstext entsteht anhand eines Templates. Dieses enthält Variablen (z.B. `$timeStamp$`). Diese werden von $ umschlossen und vor dem Versand eines `notificationList`-Elements ersetzt.

sendAlert

Operation:	`sendAlert`
Eingabe-Header:	*WS-Security (optional)*
Eingabe-Parameter:	`notificationList`
Ausgabe-Header:	*WS-Security (optional)*
Ausgabe-Parameter:	`void`

Der Web Service `sendAlert` sendet einen Alarm an die entsprechende Stelle und gehört zum Java-Teil von PegelSuite. Als Eingabe-Header kann optional ein WS-Security-Header hinzugefügt werden. Die Eingabe selbst enthält ein `notificationList`-Element (siehe Listing 4.8). Als Ausgabe-Header kann wieder ein optionaler WS-Security-Header hinzugefügt werden.

4.4. Java-Komponenten

Dieses Kapitel beschreibt den Entwurf der Java-Komponenten. Anhand des Klassendiagramms in Abbildung 4.16 werden die einzelnen Java-Klassen im Folgenden vorgestellt. Abbildung 4.16 enthält dabei lediglich die Klassen, welche Assoziationen zu Klassen anderen Pakete besitzen. Die weiteren Klassen eines Pakets werden jeweils in einer extra Abbildung dargestellt. Die entsprechenden Absätze beziehen sich jeweils auf Abbildung 4.16 und die entsprechende Abbildung des Pakets.

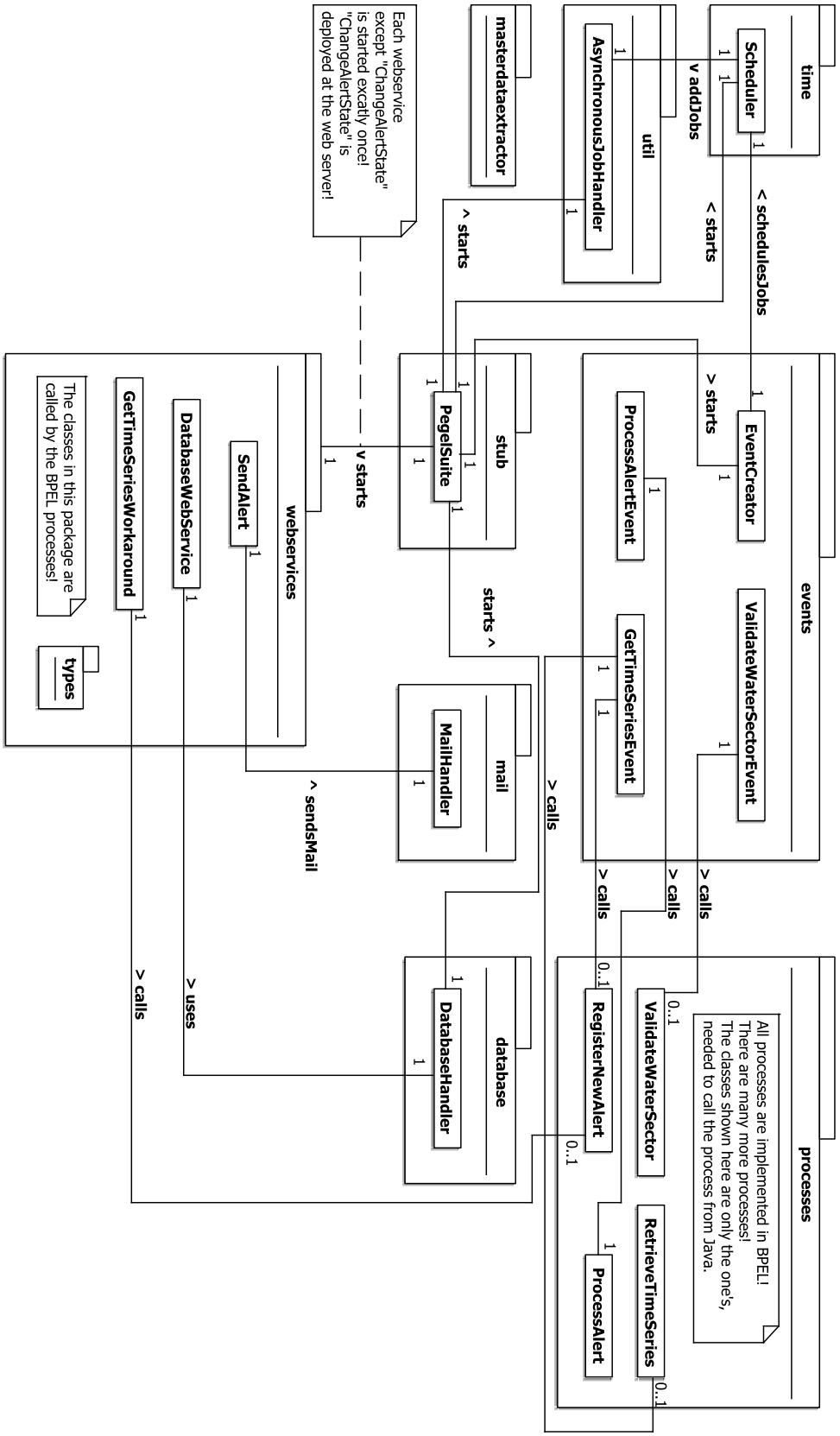


Abbildung 4.16.: Die Java-Komponenten von PegelSuite

Die Java-Komponenten von PegelSuite lassen sich grob in die Pakete `stub`, `events`, `time`, `util`, `database`, `mail`, `webservices`, `types`, `processes` und `masterdataextractor` unterteilen.

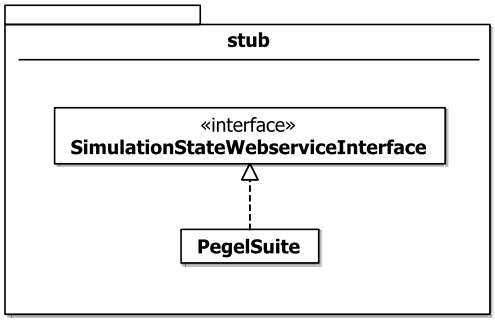


Abbildung 4.17.: Die Java-Komponenten von PegelSuite: `stub`

Im Paket `stub` befinden sich die Klasse `PegelSuite` sowie das Interface `SimulationStateWebservice`. Es enthält die Haupt-Klasse zum starten von PegelSuite. Die Klasse `PegelSuite` implementiert das Interface `SimulationStateWebservice`. Diese startet jeweils eine Instanz der Klassen `EventCreator`, `Scheduler`, `AsynchronousJobHandler` und `DatabaseHandler`. Außerdem werden alle Klassen (Web Services) des Pakets `webservice`, mit Ausnahme von `ChangeAlertState`, gestartet. Wenn eine Simulation mit PegelSim (siehe [BGH+10]) durchgeführt werden soll, wird eenfalls der Web Service `SimulationStateWebService` gestartet, welcher in der Klasse `PegelSuite` implementiert ist.

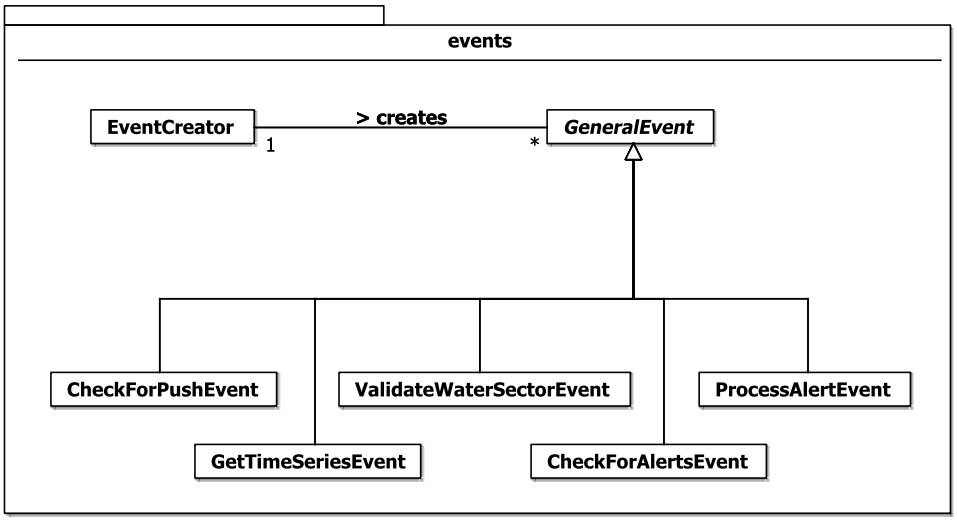


Abbildung 4.18.: Die Java-Komponenten von PegelSuite: `events`

`events` enthält die Klassen `EventCreator`, `GeneralEvent`, `CheckForPushEvent`, `GetTimeSeriesEvent`, `ValidateWaterSectorEvent`, `CheckFor-`

AlertsEvent und ProcessAlertEvent, welche die einzelnen auftretenden Events realisieren. Die Klasse EventCreator ist für das Erstellen der einzelnen Events zuständig. Diese leiten sich alle von der abstrakten Klasse GeneralEvent ab. Ein erstellter Event wird dann anschließend an den Scheduler weitergeleitet, welcher diesen zur angegebenen Zeit ausführt. CheckForPushEvent überprüft regelmäßig, ob ein erwarteter Push nicht angekommen ist. Ist dies der Fall, dann wird der Event GetTimeSeriesEvent ausgeführt. Dieser kann bei fehlenden Daten den Process RetrieveTimeSeries aufrufen. Ist der Pegel nicht erreichbar, dann kann dieser auch den BPEL-Prozess RegisterNewAlert ausführen. Der Event ValidateWaterSectorEvent überprüft in regelmäßigen Abständen ob zu plausibilisierende Daten vorliegen. Ist dies der Fall, wird der Prozess ValidateWaterGauge aufgerufen. CheckForAlertsEvent überprüft ob es aktive Alarme gibt. Wenn dies der Fall ist wird jeder Alarm durch den Event ProcessAlertEvent an den Prozess ProzessAlert weitergeleitet.

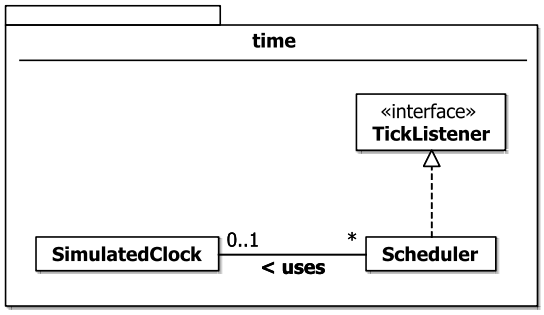


Abbildung 4.19.: Die Java-Komponenten von PegelSuite: time

Das Paket time ist für die Zeitfunktionen wie den Scheduler zuständig und enthält die Klassen SimulatedClock, Scheduler und das Interface TickListener. Der Scheduler implementiert das Interface TickListener und verwendet bei einer Simulation die Klasse SimulatedClock. Diese regelt die Zeit bei einer Simulation. Wie bereits erwähnt empfängt der Scheduler die Events vom EventCreator. Ist der Zeitpunkt der Ausführung für einen Event erreicht, gibt der Scheduler diesen Event als Job an den AsynchronousJobHandler weiter.

util enthält die Klassen Pair, AsynchronousJobHandler und Worker. Der AsynchronousJobHandler verteilt die angenommenen Jobs an seine Worker, welche diese bearbeiten. Neben diesen Klassen existiert im Paket util noch der Datentyp Pair<A,B>. Dieser wird an mehreren Stellen System verwendet und repräsentiert ein Objekt-Paar.

Das Paket database enthält für die Datenbank wichtige Klassen. Dies sind ConnectionPool und DatabaseHandler. Die wichtigste Klasse innerhalb des Pa-

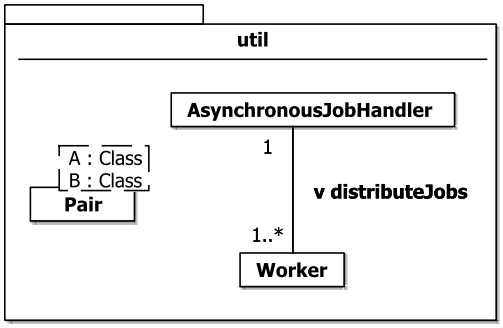


Abbildung 4.20.: Die Java-Komponenten von PegelSuite: `util`

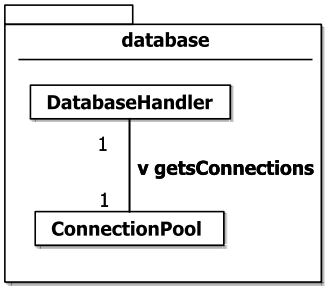


Abbildung 4.21.: Die Java-Komponenten von PegelSuite: `database`

kets ist die Klasse `DatabaseHandler`. Diese enthält alle Methoden des Systems, welche auf die Datenbank zugreifen und regelt deren Zugriff. Die einzelnen Verbindungen (entsprechen verschieden Transaktionen in Java), holt sich dieser aus einem `ConnectionPool`, welcher mehrere Verbindungen bereitstellt.

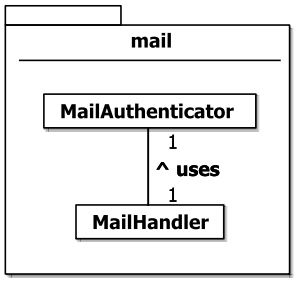


Abbildung 4.22.: Die Java-Komponenten von PegelSuite: `mail`

Für das Versenden von E-Mails existiert das Paket `mail`, welches die Klassen `MailAuthenticator` und `MailHandler` enthält. Das Versenden übernimmt die Klasse `MailHandler`, welche für die Authentifizierung die Klasse `MailAuthenticator` benötigt.

`webservices` enthält die Klassen, `DatabaseWebService`, `GetTimeSeriesWorkaround`, `ValidateWaterGauge`, `ValuePlusPlus`, `SendAlert` und `Change-`

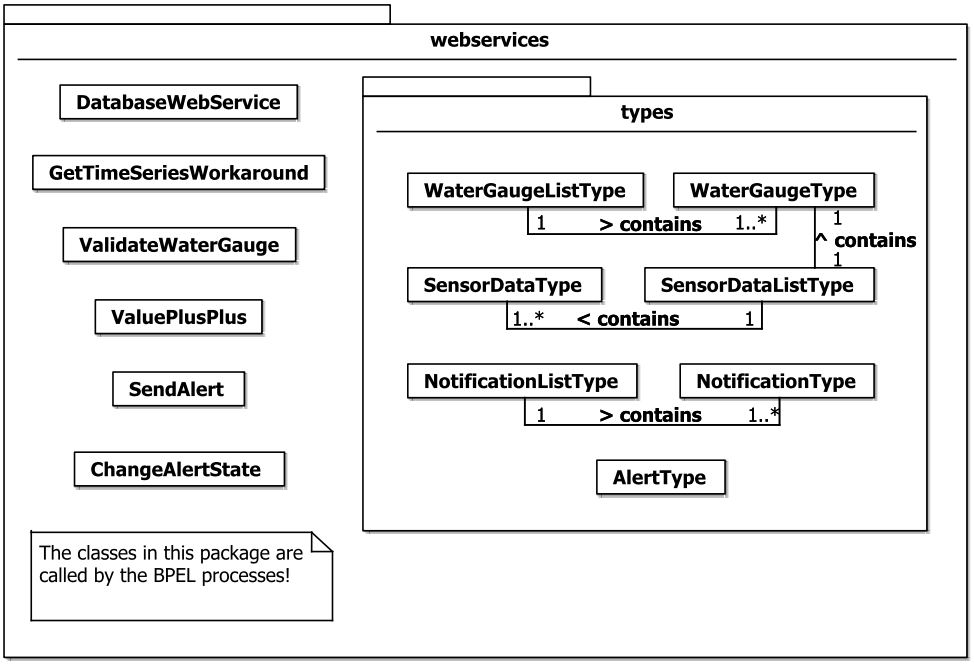


Abbildung 4.23.: Die Java-Komponenten von PegelSuite: `webservices`

`AlertState`, welche die einzelnen Web Services repräsentieren. Außerdem enthält es noch das Paket `types`. Dieses Paket wird zur Kommunikation zwischen BPEL und dem Java-Komponenten benötigt. `DatabaseWebService` enthält dabei alle Operationen, welche einen Datenbankzugriff erfordern. Diese rufen die dementsprechende Methode in der Klasse `DatabaseHandler` auf. Bei der Klasse `GetTimeSeriesWorkaround` handelt es sich um einen Web Service, welcher die richtige URL des Pegels für den Aufruf von `getTimeSeries` einsetzt. Dies ist in der verwendeten Version von Apache ODE leider nicht direkt in BPEL möglich. Ist der aufgerufene Pegel nicht erreichbar, ruft dieser Web Service den Prozess `RegisterNewAlert` auf. `ValidateWaterGauge` kümmert sich um die eigentliche Validierung indem die Fuzzy-Logik der BfG gestartet wird. Bei dem Web Service `ValuePlusPlus` handelt es sich um den Workaround für den „Integer-Double-Bug“(siehe Abschnitt 4.3.7) in Apache ODE. `SendAlert` sendet einen Alarm per E-Mail. Dies geschieht über die Klasse `MailHandler`. Bei dem Web Service `ChangeAlertState` handelt es sich um einen RESTful Web Service. Dieser kann über die HTTP-GET-Methode den Status eines Alarms ändern.

Das Paket `types` enthält die Klassen `WaterGaugeListType`, `WaterGaugeType`, `SensorDataListType`, `SensorDataType`, `NotificationListType`, `NotificationType` und `AlertType`. Diese repräsentieren die einzelnen Java-Implementationen der Datentypen, welche bereits in Abschnitt 4.3 vorgestellt wurden. Diese werden zur Kommunikation zwischen den BPEL-Prozessen und den Java-Komponenten verwendet. Ein `WaterGaugeListType`-Objekt (Liste von

Pegeln) enthält dabei mehrere `WaterGaugeType`-Objekte (Pegel). Ein `SensorDataListType`-Objekt (Liste von Sensordaten eines Pegels) enthält ein `WaterGaugeType`-Objekt und mehrere `SensorDataType` (Sensordaten). Ein Objekt der Klasse `NotificationListType` (Benachrichtigungsliste) kann mehrere `NotificationType`-Objekte (Benachrichtigungen) enthalten. `AlertType` enthält Informationen zu einem Alarm.

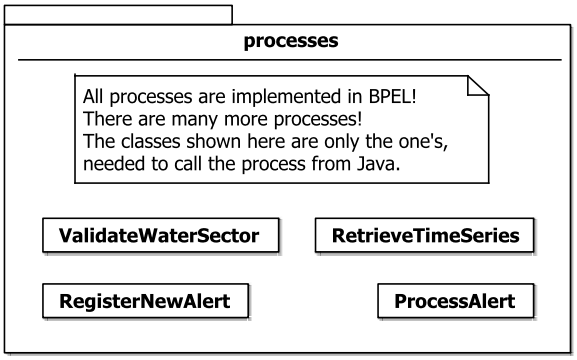


Abbildung 4.24.: Die Java-Komponenten von PegelSuite: `processes`

`processes` enthält die Java-Implementationen von BPEL-Prozessen und dient dazu, diese in BPEL aufzurufen. Im einzelnen sind dies die Klassen `ValidateWaterSector`, `RetrieveTimeSeries`, `RegisterNewAlert` und `ProcessAlert`.

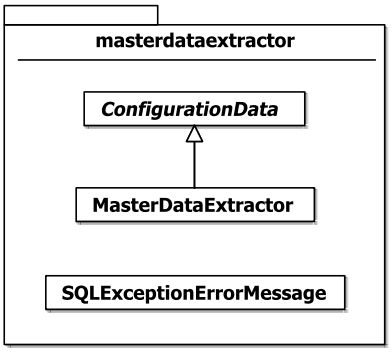


Abbildung 4.25.: Die Java-Komponenten von PegelSuite: `masterdataextractor`

Das Paket `masterdataextractor` ist unabhängig von Rest des Systems. Es enthält ein Programm, welches dazu dient die Datenbank mit den erforderlichen Daten zu füllen. Die einzelnen Klassen sind `MasterDataExtractor`, `ConfigurationData` und `SQLExceptionErrorMessage`. Die Hauptklasse des Pakets ist `MasterDataExtractor`. Diese erweitert die abstrakte Klasse `ConfigurationData` und enthält lediglich die Daten, welche eingetragen werden sollen, sowie die Datenbank-Zugangsdaten. Die zugehörigen Methoden wurden

in `MasterDataExtractor` implementiert. Die Klasse `SQLExceptionErrorMessage` enthält Fehlermeldungen, welche bei dem Füllen der Datenbank auftreten könnten.

4.5. Probleme bei der Implementierung

Der in Abbildung 4.8 dargestellte Ablauf der Workflows ließ sich leider nicht problemlos in BPEL realisieren. Bei der Implementation traten durch die Verwendung von BPEL und die Apache ODE einige Probleme auf, die im Folgenden erläutert werden.

Die Entscheidung, die Datenbank an eine Java-Komponente anzubinden führte innerhalb der Workflows zu vielen unnötigen Web Services, die ebenso viel überflüssigen Traffic verursachen. Darunter leidet insbesondere die Performance des Systems. Wenn man sich die in den früheren Abschnitten (insbesondere Abschnitt 4.3) dargestellten Web Services anschaut, stellt man schnell fest, dass die Hauptaufgabe der Prozesse in `assign`-Anweisungen resultiert. Dies entspricht in den meisten Fällen lediglich einer einfachen Variablenzuweisung in Java. Der Unterschied ist, dass die Zuweisung innerhalb von BPEL wesentlich komplexer ist und dadurch bei der Implementation leichter Fehler gemacht werden können.

Aufgrund der veralteten Dokumentation von Apache ODE war es ebenfalls nicht möglich, WS-Security zu verwenden.

Weiterhin machte es Probleme die original WSDL-Dateien zu publizieren. In der Grundeinstellung generiert Apache ODE eine eigene WSDL-Datei für die einzelnen Prozesse anstatt, die original WSDL-Datei zu verwenden, wie man es eigentlich erwarten würde. Dabei wurden von ODE insbesondere die Namen der Services verändert. Da von der BfG allerdings die WSDL-Dateien für die Services `getTimeSeries` und `putTimeSeries` bereits vorgegeben waren, mussten die Original-WSDL-Dateien verwendet werden. Dies geschieht durch das Erstellen einer *<Servicename>*`.axis2`-Datei, mit dem Inhalt aus Listing 4.10 im Deploy-Verzeichnis. *<Servicename>* bezeichnet dabei den Namen des Services innerhalb der WSDL-Datei.

```
1 <service>
2  <parameter name="useOriginalwsdl">true</parameter>
3 </service>
```

Listing 4.10: Inhalt einer AXIS2-Datei

Ein weiteres Problem bestand in der Benutzung von Schemas, innerhalb eines BPEL-Prozesses, mit einer nicht-lokalen `schemaLocation` (`http://...`). Diese werden von Apache ODE noch nicht unterstützt. Auch die verwendete Version

2.0 Beta2 gibt während dem Deploy-Vorgang viele Fehler aus die dieses Problem betreffen. Die Lauffähigkeit wird dadurch jedoch nicht beeinträchtigt. Die Version 2.0 Beta2 wird höchstwahrscheinlich jedoch nicht zu Version 2.0 weiterentwickelt. Stattdessen soll diese auf Version 1.3.x basieren.

Weiterhin ist Apache ODE nicht sehr stabil. Der Deploy-Vorgang dauert, aufgrund der Tatsache, dass jedes externe Schema geladen und validiert wird, sehr lange (ca. 30 Minuten). Dabei kann es vorkommen, dass eine der zu validierenden Schemas nicht erreichbar ist und ODE daher den Deploy-Vorgang mit einem Fehler abbricht. Dies war während einiger Tests z.B. bei der `xml.dtd` des W3Cs der Fall. Die Berechnung von Integer-Zahlen mittels XPath gibt, ungeachtet des angegebenen Datentyps, einen Double-Wert zurück. Dies lässt sich auf einen Bug in Apache ODE zurückführen. Die Verwendung einer dynamischen IP-Adresse für den Aufruf eines Web Services ist in ODE auch noch nicht möglich. Für diese beiden Probleme wurde ein Workaround in Java geschrieben (siehe `valuePlusPlus` und `getTimeSeriesWorkaround` in Abbildung 4.23). Auch während der Laufzeit von ODE kommt es häufiger zu einem kurzen Freeze bei der Verarbeitung der Daten und zu Fehlern, die z.B. einen fehlerhaften Zugriff auf die ODE-interne Datenbank zurückzuführen sind.

5. Benutzerhandbuch

Im Folgenden wird die Installation, die Konfiguration, der Betrieb und die Wartung von PegelSuite beschrieben. Zuvor wird jedoch die benötigte Software und die verwendeten Standards aufgelistet. Die Einrichtung von *PegelSim* zur Durchführung einer Simulation ist nicht Teil des Benutzerhandbuchs. Diese kann in [BGH+10] nachgeschlagen werden.

5.1. Verwendete Software und Standards

Für den Betrieb und die Wartung von PegelSuite wird eine Reihe von Software benötigt. Dabei spielt auch die Versionsnummer der Software eine große Rolle. Diese wird in Tabelle 5.1 zusammen mit den verwendeten Standards aufgelistet.

Name	**Version**
Java SE JDK	6
Eclipse IDE für JavaEE	3.4.2 (Ganymede)
BPEL-Designer (Eclipse-Plugin)	0.4.0
Apache Tomcat	7.0.0
Apache ODE	2.0 Beta 2
MySQL	5.0.26
WS-BPEL	2.0
WSDL	1.1
SOAP	1.1
XML Schema	1.0
XPath	1.0

Tabelle 5.1.: Verwendete Software und Standards mit Versionsnummer

5.1.1. Java, Eclipse und der BPEL-Designer

Das System wurde mittels Java SE SDK 6[1] implementiert. Als Entwicklungs-IDE wurde Eclipse für JavaEE 3.4.2 (Ganymede)[2], zusammen mit dem Plugin BPEL-Designer 0.4.0 [3], verwendet. Die Verwendung dieser Versionen wird empfohlen, aber es können auch neuere Versionen, soweit diese untereinander kompatibel sind, verwendet werden.

5.1.2. Apache Tomcat und ODE

Bei der Verwendung des Webservers Apache ODE[4] zusammen mit der BPEL-Engine Apache ODE[5] funktioniert das Zusammenspiel nur mit den angegebenen Versionen.

5.1.3. MySQL

Die Version der verwendeten MySQL-Datenbank[6] war 5.0.26. Es sollten aber auch höhere Versionen verwendbar sein.

5.1.4. XHydro

Innerhalb von PegelSuite wurde Version 1.1 des XHydro-Standards implementiert. XHydro selbst dient zur Übertragung von Zeitreihen. Aufgrund der Komplexität dieses Standards wurde dieser nur als Schnittstelle zwischen Pegel und PegelSuite, nicht aber innerhalb von PegelSuite verwendet.

5.1.5. WS-BPEL und die restlichen Standards

Der Standard WS-BPEL wurde in Version 2.0 verwendet. Er dient der Orchestrierung von Web Services. Innerhalb von BPEL können die Standards WSDL 1.1, SOAP 1.1, XML Schema 1.0 und XPath 1.0 in genau diesen Versionen verwendet werden.

5.2. Installation und Konfiguration

In diesem Abschnitt wird die Installation und Konfiguration von Apache Tomcat, Apache ODE und PegelSim beschrieben.

[1] `http://www.oracle.com/technetwork/java/index.html` (abgerufen: 16.07.2011)
[2] `http://www.eclipse.org` (abgerufen: 16.07.2011)
[3] `http://www.eclipse.org/bpel` (abgerufen: 16.07.2011)
[4] `http://tomcat.apache.org` (abgerufen: 16.07.2011)
[5] `http://ode.apache.org` (abgerufen: 16.07.2011)
[6] `http://www.mysql.com` (abgerufen: 16.07.2011)

5.2.1. Apache Tomcat

Die heruntergeladene ZIP-Datei kann einfach in ein beliebiges Verzeichnis entpackt werden. Anschließend muss für die Konfiguration von Tomcat ein neuer Benutzer in der Datei `<tomcat_home>/conf/tomcat-users.xml` angelegt werden (siehe Listing 5.1). Für den Betrieb ist es allerdings nicht nötig Tomcat zu konfigurieren.

```
1 <tomcat-users>
2  <user username="tomcat" password="secret"
3   roles="standard,manager-gui,admin-gui" />
4 </tomcat-users>
```

Listing 5.1: Apache Tomcat: Anlegen eines neuen Benutzers

5.2.2. Apache ODE

Aus der heruntergeladenen ZIP-Datei muss die Datei `ode.war` in das Verzeichnis `<tomcat_home>/webapps` entpackt werden. ODE installiert sich beim nächsten Start von Apache Tomcat. Dies geschieht durch den Befehl `catalina run` im Verzeichnis `<tomcat_home>/bin`.

5.2.3. PegelSuite

Die Installation und Konfiguration von PegelSuite gestaltet sich aufgrund der prototypischen Beschaffenheit etwas schwieriger. Als erster Schritt sollte die Datenbank erstellt und mit Daten gefüllt werden. Anschließend kann PegelSuite konfiguriert werden.

Datenbank anlegen

Zuerst muss das Datenbank-Schema angelegt werden. Dies geschieht durch Ausführung des SQL-Scripts `src/database/createpegelsuiteschema.sql`. Anschließend kann die Tabelle mit Daten gefüllt werden. Da die Datenbank auch einige Konfigurationsdaten, wie Personen, Zuständigkeiten, `getTimeSeries`-Adressen der Pegel etc. enthält, muss dies manuell geschehen. Wenn lediglich eine Simulation durchgeführt werden soll, kann das Tool `MasterDataExtractor` (Projekt `src/pegelSuite`) verwendet werden. Dieses liest aus der PegelSim-Datenbank die benötigten Informationen und speichert sie in der PegelSuite-Datenbank. Weitere benötigte Werte können in der Klasse `ConfigurationData` verändert werden. Dort lassen sich auch die Zugangsdaten der beiden Datenbanken (PegelSuite und PegelSim) konfigurieren. Durch das Aufführen der Java-Klasse `MasterDataExtractor` (in Eclipse: Rechtsklick auf die Klasse → „Run As“ → „Java Application“) werden die Werte übertragen.

Konfiguration von PegelSuite

Java-Variablen

Wie bereits auch bei dem Tool `MasterDataExtractor` muss die Konfiguration von PegelSuite (Projekt `src/pegelSuite`) innerhalb der Java-Klassen geschehen. Dies geschieht durch Klassenvariablen. Es handelt sich dabei immer um Konstanten, deren Namen zur Unterscheidung von anderen Klassenvariablen zusätzlich komplett in Großbuchstaben geschrieben sind. Die meisten Einstellungen lassen sich über die Klasse `PegelSuite` vornehmen, jedoch müssen vereinzelt auch an anderen Stellen Variablen verändert werden.

Innerhalb der Klasse PegelSuite lassen sich die Adressen der Web Services festlegen, die in Java implementiert wurden. Weiterhin lässt sich festlegen, ob es sich bei der Ausführung um eine Simulation handelt, ob die Ausführung detaillierte Informationen ausgibt und wie hoch die Anzahl der Worker-Threads, die der `AsynchronousJobHandler` zur Verfügung hat, ist.

In der Klasse `DatebaseHandler` sind die Datenbankeinstellungen vorzunehmen. Weiterhin ist in dieser Klasse das Offset für die Berechnung des Wertes `toTimeStamp` der Datenbanktabelle `MissingData` angegeben, welcher von mehreren Methoden der Klasse verwendet wird.

Die Klasse `MailHandler` enthält die Einstellungen die für das Versenden einer E-Mail nötig sind.

`ValidateWaterSectorEvent` besitzt die Variable `EQUIDISTANCE_OF_EXECUTIONS`, welche den Abstand zwischen den Ausführungen des Events angibt.

BPEL-Prozesse

Die URLs der Web Services im Paket `processes` müssen ebenfalls manuell eingestellt werden, da eine dynamische Zuweisung durch Apache ODE nicht möglich ist. Um diese Einstellung vorzunehmen, muss das Attribut `location` des Elements `soap:address` (Nachfolger von `service`) innerhalb der jeweiligen WSDL-Datei verändert werden.

ChangeAlertState

Der Web Service `ChangeAlertState` wurde nicht, wie der Rest des Systems im Projekt `src/pegelSuite` angelegt, sondern befindet ist im Projekt `src/rest`. Die Auslagerung war nötig, um die WAR-Datei, die dieses Projekt erzeugt, nicht unnötig groß werden zu lassen, indem diese unnötige Klassen enthält. Da dieses Projekt in der Implementierung komplett unabhängig von `src/pegelSuite` ist, müssen innerhalb der Klasse `ChangeAlertState` erneut die Datenbankeinstellungen vorgenommen werden. Nachdem die Konfiguration des Projekts abgeschlossen ist, muss die WAR-Datei zum Deployen erstellt werden. Dies geschieht in Eclipse durch einen Rechtsklick auf das Projekt → „Export“ → „WAR file“. Wie auch die Datei `ode.war` sollte diese Datei ins Verzeichnis `<tomcat_home>/webapps` kopiert werden.

Nachdem Tomcat das erste mal gestartet wurde, existiert das Verzeichnis `<tomcat_home>/webapps/ode/WEB-INF/processes`. In dieses muss, bei beendetem Web Server, das Paket `processes` aus dem Projekt `src/pegelSuite` kopiert werden. Es müssen lediglich der Ordner und alle direkt in diesem liegenden Dateien kopiert werden. Das Kopieren der Unterordner ist nicht nötig.

5.3. Betrieb

Nachdem die gesamte Konfiguration abgeschlossen wurde, kann PegelSuite gestartet werden. Hierbei ist die Reihenfolge sehr wichtig. Zuerst sollten die Java-Komponenten gestartet werden, da die WSDL-Dateien der Web Services beim Start von Apache ODE bereits publiziert sein müssen. Dies geschieht in Eclipse durch einen Rechtsklick auf die Klasse `PegelSuite` → „Run As“ → „Java Application“. Anschließend kann durch den Befehl `catalina run` im Verzeichnis `<tomcat_home>/bin` Apache Tomcat und Apache ODE gestartet werden. Der anschließende Deploy-Vorgang nimmt einige Zeit (im Test ca. 30 Minuten) in Anspruch. Der Grund für die lange Deploy-Zeit besteht darin, das ODE jede WSDL-Datei und jedes Schema läd und validiert. Bevor PegelSuite verwendet werden kann, muss der Deploy-Vorgang komplett abgeschlossen sein.

5.4. Wartung

Sobald PegelSuite gestartet wurde, läuft das System autonom und es bedarf nur in Alarmfällen einer Regulierung durch die zuständigen Personen. Diese müssen den Alarm bearbeiten, dessen Status ändern und evtl. durch den Alarm fehlende Daten in der Datenbank ergänzen.

6. Zusammenfassung und Erfahrungen

Diese Diplomarbeit hatte das Ziel ein prototypisches System zur automatisierten Verarbeitung von Pegeldaten auf der Basis von Web Services zu entwickeln. Die Verwendung gängiger Standards spielte dabei eine große Rolle. Insbesondere die Nutzung von WS-BPEL 2.0 zur Orchestration der Web Services und XHydro als Austauschformat zwischen diesen, waren zwei der Haupt-Anforderungen. Zudem sollte das System performant und skalierbar sein.

Die Verarbeitung der Pegeldaten lässt sich in 4 Workflows einteilen (siehe Abschnitt 4.3). Der 1. Workflow ist für den Datenempfang zuständig. Der Empfang der Daten kann durch einen Push vom Pegel geschehen oder die Abfrage der Daten vom Pegel geschehen. Nach dem Empfang der Daten wird für diese im 2.Workflow ein Qualitätsmerkmal berechnet und zusammen mit den Daten in einer Datenbank, als Rohdaten, gespeichert. Der 3. Workflow plausibilisiert diese Daten in einem vorgegebenen Intervall. Sind diese plausibel, so werden die Rohdaten in der Datenbank zusätzlich als plausibilisierte Daten gespeichert. Sollte bei der Ausführung dieser 3 Workflows etwas schief gehen, wird ein Alarm ausgelöst. Für die Bearbeitung von Alarmen ist der 4. Workflow zuständig. Ein aufgetretener Alarm wird in der Datenbank gespeichert und in regelmäßigen Abständen eine Benachrichtigung per E-Mail versandt. Nach einigen Benachrichtigungen ohne Reaktion (Statusänderung) auf den Alarm, werden die Personen der nächst höheren Eskalationsstufe benachrichtigt. Die Bearbeitung eines Alarm erfolgt manuell durch die dementsprechenden Sachbearbeiter.

Bei der Verwendung des Standards XHydro (siehe Abschnitt 2.5) zeigte sich während der Entwicklung, dass dieser für die Aufgabe der Übertragung zu komplex ist. Durch die teilweise unnötige Verschachtelung von Tags, die lediglich dem Hinzufügen von Erweiterungselementen dient, entsteht ein großer Overhead bei der Übertragung und macht die Verarbeitung zudem unnötig kompliziert. Die sogenannten Joker-Elemente, lassen sich zudem erst durch eine Anpassung des Codes verwenden. Besser wäre es gewesen, nicht einen allgemeinen, erweiterbaren Standard, sondern eher einen kompakten, genau auf sein Aufgabengebiet zugeschnittenen Standard, ohne besonderes Augenmerk auf die Erweiterbarkeit, zu entwickeln. Die Idee der Erweiterbarkeit ist zwar in Ordnung, jedoch zerstört dessen gewählte Realisierung den Grundgedanken hinter XHydro.

Die Verwendung von Abkürzungen als Namen für die XML-Elemente von XHydro entspricht zwar nicht ganz dem Ziel von XML menschenlesbar zu sein, ist allerdings eine gute Entscheidung, da es sich um ein Übertragungsformat handelt und dementsprechend die Kompaktheit der Daten im Vordergrund stehen sollte.

Eine einfache Verarbeitung der Daten, durch eine überschaubare Struktur des Formats, würde durch eine einfachere und schnellere Verarbeitung der Daten in einem System die Performanz erhöhen. Dies ist ein wichtiger Faktor, wenn man bedenkt, dass allein die WSV bereits rund 1450 Pegel unterhält. Wenn man annimmt, dass jeder dieser Pegel allein für die Wasserstandsrohdaten, welche von PegelSuite verarbeitet werden, 2 Sensoren besitzt, die alle 15 Minuten einen Wert liefern, dann darf die vollständige Verarbeitung eines Wertes maximal 310 ms in Anspruch nehmen. Messungen haben ergeben, dass PegelSuite bereits für den Durchlauf des 1. Workflows für einen Wert zwischen 700 ms und 1200 ms benötigt. Das das System damit bereits überfordert ist, muss nicht erwähnt werden und dies würde sich nochmals verschlimmern, wenn zusätzlich noch Werte wie die Wassertemperatur oder der PH-Wert hinzukommen.

Der Wert `locationNumber` sollte XHydro noch hinzugefügt werden um eine eindeutige Identifikation des Pegels zu gewährleisten. Dies ist durch den `locationName` nicht gegeben. In PegelSuite wurde die `locationNumber` in XHydro in einer Austauschnummer (`xid`-Element) untergebracht.

Der Grund für die langsame Verarbeitung der Daten in PegelSuite ist allerdings nicht auf XHydro, sondern auf die Verwendung von WS-BPEL (siehe Abschnitt 2.4) zurückzuführen. Durch unnötige Web Services, die zur Kommunikation mit der Datenbank benötigt werden, wird viel unnötige Zeit in die Kommunikation zwischen BPEL-Prozessen und Java-Komponenten investiert.

Die Verwendung von BPEL war eine der wichtigsten Anforderungen in dieser Diplomarbeit und stellte sich schnell als die größte Fehlentscheidung heraus (siehe Abschnitt 4.5). Nach anfänglichen Schwierigkeiten eine geeignete freie BPEL-Engine zu finden, konnte sich Apache ODE 2.0 Beta2 (wird nicht weiterentwickelt zu Version 2.0) als geeignete Implementierung durchsetzen. Andere Versionen von ODE waren ebenfalls nicht lauffähig. Dies lag an mehreren Faktoren:

Einige Versionen/BPEL-Engines unterstützen die Verwendung der originalen WSDL nicht. Dies war allerdings aufgrund der vorgegebenen WSDL-Dateien für die Web Services `getTimeSeries` und `putTimeSeries` erforderlich.

Weiterhin wurde die Verwendung nicht-lokaler `schemaLocations` innerhalb einer WSDL-Datei nicht unterstützt. In Apache ODE 2.0 Beta2 werden zwar dies-

bezüglich auch viele Fehler ausgegeben, was der Lauffähigkeit der Prozesse allerdings nicht im Wege steht.

Generell lässt sich sagen, das bei Apache ODE die hohe Versionsnummer leider nicht auf einen hohen Grad von unterstützen Funktionen oder Fehlerfreiheit schließen lässt. Bei dem langsamen Deploy-Vorgang kann es vorkommen, dass eine der zu validierenden Schemas nicht erreichbar ist und ODE daher den Deploy-Vorgang abbricht. Dies war während einiger Tests z.B. bei der `xml.dtd` des W3Cs der Fall. Auch der Fehler, dass eine Berechnung von Integer-Zahlen mittels XPath, ungeachtet des angegebenen Datentyps, einen Double-Wert zurück liefert, lässt sich auf Apache ODE zurückführen. Die Verwendung einer dynamischen IP-Adresse für den Aufruf eines Web Services ist in ODE nicht möglich. Auch während der Laufzeit von ODE kommt es häufiger zu einem kurzen Freeze bei der Verarbeitung der Daten und zu Fehlern, die z.B. einen fehlerhaften Zugriff auf die ODE-interne Datenbank zurückzuführen sind.

Weitere Probleme bei der Entwicklung ließen sich auf die mangelhafte und veraltete Dokumentation von Apache ODE zurückführen. Wegen dieser war es z.B. nicht möglich, WS-Security zu verwenden. Datenbankzugriffe konnten zwar innerhalb von BPEL direkt erfolgen, jedoch war diese Funktion auf die Ausgabe einer Zeile beschränkt. Dies war für PegelSuite jedoch unzureichend. Somit mussten alle Datenbankzugriffe innerhalb von Java realisiert werden. Dies wiederum führte zu der bereits erwähnten unnötigen Menge an Web Services. BPEL selbst besitzt dadurch im Grunde nur noch die Aufgabe einige Schleifen, if-Abfragen und Variablen-Zuweisungen zu tätigen. Sofern es nicht durch den „Integer-Double-Bug“ unmöglich ist, können zusätzlich noch einige kleine Berechnungen mit XPath durchgeführt werden.

Zu BPEL selbst kann man noch anmerken, dass eine Sprache im XML-Stil nicht gerade übersichtlich ist. Durch Tools wie den BPEL-Designer, wird dies zwar in einem gewissen Maße wieder ausgeglichen, allerdings sind noch immer einige Änderungen direkt im Code nötig und XML ist in dieser Hinsicht nicht so übersichtlich in der Struktur wie etwa Java. In BPEL wäre es wünschenswert, anstatt der bestehenden Unterstützung von XPath 1.0 eine Unterstützung der aktuellen Version XPath 2.0 zu haben. Im Vergleich zu Version 1.0 besitzt XPath 2.0 einige neue, nützliche Funktionen.

Im Grunde lassen sich alle Funktionen, die BPEL anbietet in einer beliebigen Programmiersprache realisieren. Die Definition eines speziellen Standards für die Orchestration von Web Services ist daher eigentlich unnötig. Es hätte vollkommen ausgereicht, die Beschreibungssprache für die Web Service Schnittstellen zu standardisieren, was durch WSDL bereits getan wird. Wenn BPEL für die Orchestrierung der Web Services verwendet wird und dennoch eine weitere Sprache

für zusätzliche Funktionen benötigt wird, so bedingt dies zusätzliche Schnittstellen an denn Fehler auftreten können. In diesem Fall sollte besser auf BPEL verzichtet werden.

Die Verwendung von BPEL eignet sich - aufgrund der zuvor genannten Defizite - nur für grobgranulare Orchestrierungen. Die Orchestrierung in PegelSuite wurde jedoch sehr feingranular vorgenommen. Hätte man jeden Workflow in PegelSuite als einen Web Service realisiert und betrachtet, so hätte man diese u.U. mit BPEL orchestrieren können. Die Services sind jedoch in diesem Fall nicht zusammenhängend genug (Workflow 3 wird z.B. durch einen Timer und nicht nach Workflow 2 gestartet), dass eine Orchestrierung auf auf diesem Level nicht sinnvoll ist.

Zusammenfassend lässt sich sagen, dass Standards wie Web Services für organisationsübergreifende Projekte nötig sind. Allerdings sollte man nicht in einen „Standardisierungsrausch“ verfallen und alles bis ins letzte Detail standardisieren, wie es bei WS-BPEL der Fall war. Es wäre ausreichend gewesen, die Schnittstelle (WSDL), das Übertragungsformat (z.B. XHydro) und die Kommunikation (HTTP) zu standardisieren. Wie die restliche interne Implementation aussieht, sollte nicht weiter reglementiert werden.Insbesondere die Sprache, in der die Web Services implementiert werden, sollte daher frei wählbar bleiben.

Der Versuch mit PegelSuite ein System zu schaffen, welches auf BPEL aufbaut und auf Anforderungen wie Performanz, Skalierbarkeit und Standardkonformität beruht, ließ sich nicht in allen Punkten umsetzen. Durch eine zu starke Nutzung von BPEL wurde das System extrem in seiner Performanz beeinträchtigt und ist somit im realen Betrieb nicht einsetzbar. Weiterhin wurden durch die vielen benötigten Schnittstellen weitere Fehlerquellen eingeführt und viele einfache Aufgaben unnötig verkompliziert.

Insgesamt läuft der Prototyp von PegelSuite sehr langsam und leider auch nicht sehr stabil. Die Instabilität wurde auf der einen Seite durch Apache ODE und auf der anderen Seite auch durch Fehler in PegelSuite selbst hervorgerufen. Aufgrund der häufigen Abstürze und der langen Deploy-Zeit von Apache ODE war es nicht möglich ausführliche Tests durchzuführen. Insbesondere der 3. Workflow konnte nicht ausreichend getestet werden, da ODE zu Beginn des Workflows, aus ungeklärten Gründen eine `SOAPFaultException` verursacht und der Workflow dadurch abgebrochen wird. Um weitere Tests diesbezüglich durchzuführen fehlte leider die Zeit in dieser Diplomarbeit.

Unter dem Aspekt der produktiven Nutzung hätte die Performanz und die Stabilität des Systems hätte als wichtiger erachtet werden sollen, wie die Verwendung von BPEL. Der wissenschaftliche, forschende Anspruch dieser Diplomarbeit war

es jedoch, die Machbarkeit unter Verwendung von BPEL sowie dessen Nutzen auszuloten. Für den realen Einsatz wäre jedoch eine Neuentwicklung von Pegel-Suite ohne die Verwendung von BPEL nötig.

A. Web Services

A.1. WSDL-Beispiel

```
<?xml version="1.0"?>
<definitions name="WaterGaugeData"
 targetNamespace="http://www.pegel-kaub.de/watergaugedata.wsdl"
 xmlns:tns="http://www.pegel-kaub.de/watergaugedata.wsdl"
 xmlns:xsd1="http://www.pegel-kaub.de/watergaugedata.xsd"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <types>
  <schema
   targetNamespace="http://www.pegel-kaub.de/watergaugedata.xsd"
   xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="Header">
    <complexType>
     <all>
      <element name="Duration" type="integer" />
     </all>
    </complexType>
   </element>
   <element name="WaterGaugeDataRequest">
    <complexType>
     <all>
      <element name="time" type="dateTime" />
     </all>
    </complexType>
   </element>
   <element name="WaterGaugeData">
    <complexType>
     <all>
      <element name="value" type="float" />
      <element name="unit" type="string" />
     </all>
    </complexType>
   </element>
  </schema>
```

```
</types>

<message name="Header">
 <part name="header" element="xsd1:Header" />
</message>
<message name="GetWaterGaugeDataInput">
 <part name="body" element="xsd1:WaterGaugeDataRequest" />
</message>
<message name="GetWaterGaugeDataOutput">
 <part name="body" element="xsd1:WaterGaugeData" />
</message>

<portType name="WaterGaugeDataPortType">
 <operation name="GetWaterGaugeData">
  <input message="tns:GetWaterGaugeDataInput" />
  <output message="tns:GetWaterGaugeDataOutput" />
 </operation>
</portType>

<binding name="WaterGaugeDataSoapBinding"
 type="tns:WaterGaugeDataPortType">
 <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
 <operation name="GetWaterGaugeData">
  <soap:operation
   soapAction="http://www.pegel-kaub.de/getwatergaugedata" />
  <input>
   <soap:body use="literal" />
  </input>
  <output>
   <soap:header message="tns:Header" part="Duration"
    use="literal" />
   <soap:body use="literal" />
  </output>
 </operation>
</binding>

<service name="WaterGaugeDataService">
 <documentation>My first service</documentation>
 <port name="WaterGaugeDataPort"
  binding="tns:WaterGaugeDataSoapBinding">
  <soap:address
   location="http://www.pegel-kaub.de/watergaugedata" />
 </port>
</service>

```

```
82 </definitions>
```

Listing A.1: WSDL-Beispiel

B. WS-BPEL

B.1. WS-BPEL-Beispiel

B.1.1. simpleInvoke.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions name="simpleInvoke"
 targetNamespace=
  "http://www.pegelsuite.de/prototypes/simpleInvoke"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
 xmlns:tns="http://www.pegelsuite.de/prototypes/simpleInvoke"
 xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
 xmlns:wsdl="http://xhydro.org/tns"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:resp="http://xhydro.org/minimal/2009/09">

 <import location="getTimeSeries.wsdl"
  namespace="http://xhydro.org/tns" />

 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
   attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   targetNamespace=
    "http://www.pegelsuite.de/prototypes/simpleInvoke">
   <element name="simpleInvokeRequest">
    <complexType>
     <sequence>
      <element name="input" type="boolean" />
     </sequence>
    </complexType>
   </element>
  </schema>
 </types>

 <message name="simpleInvokeRequestMessage">
  <part element="tns:simpleInvokeRequest"
```

```
34    name="simpleInvokeRequestPayload" />
35  </message>
36  <message name="simpleInvokeResponseMessage">
37   <part element="resp:tsel" name="simpleInvokeResponseTsel" />
38  </message>
39
40  <portType name="simpleInvokePT">
41   <operation name="simpleInvokeOperation">
42    <input message="tns:simpleInvokeRequestMessage" />
43    <output message="tns:simpleInvokeResponseMessage" />
44   </operation>
45  </portType>
46
47  <plnk:partnerLinkType name="simpleInvokePLT">
48   <plnk:role name="simpleInvokeProvider"
49    portType="tns:simpleInvokePT" />
50  </plnk:partnerLinkType>
51  <plnk:partnerLinkType name="getTimeSeriesPLT">
52   <plnk:role name="getTimeSeriesProvider"
53    portType="wsdl:getTimeSeries_WebService" />
54  </plnk:partnerLinkType>
55
56  <binding name="simpleInvokeBinding" type="tns:simpleInvokePT">
57   <soap:binding style="document"
58    transport="http://schemas.xmlsoap.org/soap/http" />
59   <operation name="simpleInvokeOperation">
60    <soap:operation
61     soapAction=
62      "http://www.pegelsuite.de/prototypes/
63       simpleInvoke/simpleInvokeOperation" />
64    <input>
65     <soap:body use="literal" />
66    </input>
67    <output>
68     <soap:body use="literal" />
69    </output>
70   </operation>
71  </binding>
72
73  <service name="simpleInvokeService">
74   <port name="simpleInvokePort"
75    binding="tns:simpleInvokeBinding">
76    <soap:address
77     location="http://localhost:8080/ode/processes/simpleInvoke"
78    />
79   </port>
```

```
80  </service>
81
82  </definitions>
```

Listing B.1: WS-BPEL-Beispiel: `simpleInvoke.wsdl`

B.1.2. `simpleInvoke.bpel`

```
1   <bpel:process name="simpleInvoke" suppressJoinFailure="yes"
2    targetNamespace=
3     "http://www.pegelsuite.de/prototypes/simpleInvoke"
4    xmlns:tns="http://www.pegelsuite.de/prototypes/simpleInvoke"
5    xmlns:bpel=
6     "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
7    xmlns:xhydrotns="http://xhydro.org/tns"
8    xmlns:xhydroreq="http://xhydro.org/ws/request">
9
10   <bpel:import location="getTimeSeries.wsdl"
11    namespace="http://xhydro.org/tns"
12    importType="http://schemas.xmlsoap.org/wsdl/" />
13   <bpel:import location="simpleInvoke.wsdl"
14    namespace="http://www.pegelsuite.de/prototypes/simpleInvoke"
15    importType="http://schemas.xmlsoap.org/wsdl/" />
16
17   <bpel:partnerLinks>
18    <bpel:partnerLink name="simpleInvokePL"
19     partnerLinkType="tns:simpleInvokePLT"
20     myRole="simpleInvokeProvider" />
21    <bpel:partnerLink name="getTimeSeriesPL"
22     partnerLinkType="tns:getTimeSeriesPLT"
23     partnerRole="getTimeSeriesProvider" />
24   </bpel:partnerLinks>
25
26   <bpel:variables>
27    <bpel:variable name="simpleInvokeRequest"
28     messageType="tns:simpleInvokeRequestMessage" />
29    <bpel:variable name="simpleInvokeResponse"
30     messageType="tns:simpleInvokeResponseMessage" />
31    <bpel:variable name="getTimeSeriesRequest"
32     messageType="xhydrotns:getTimeSeriesRequestMessage" />
33    <bpel:variable name="getTimeSeriesResponse"
34     messageType="xhydrotns:getTimeSeries_ResponseMessage" />
35   </bpel:variables>
36
37   <bpel:sequence name="mainSequence">
38    <bpel:receive name="receiveInput"
```

```
39    partnerLink="simpleInvokePL" portType="tns:simpleInvokePT"
40    operation="simpleInvokeOperation"
41    variable="simpleInvokeRequest" createInstance="yes" />
42   <bpel:assign validate="no" name="assignInvokeInput">
43    <bpel:copy>
44     <bpel:from>
45      <bpel:literal xml:space="preserve">
46       <tns:getTimeSeriesRequest
47        xmlns:tns="http://xhydro.org/ws/request"
48        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
49        <tns:locationNumber>25100107</tns:locationNumber>
50        <tns:locationName>Mainz</tns:locationName>
51        <tns:parameterCode>W</tns:parameterCode>
52        <tns:from>2010-03-31T14:00:00+02:00</tns:from>
53        <tns:to>2010-03-31T14:15:00+02:00</tns:to>
54       </tns:getTimeSeriesRequest>
55      </bpel:literal>
56     </bpel:from>
57     <bpel:to variable="getTimeSeriesRequest"
58      part="getTimeSeriesRequestPart" />
59    </bpel:copy>
60   </bpel:assign>
61   <bpel:invoke name="invokeGetTimeSeries"
62    partnerLink="getTimeSeriesPL"
63    portType="xhydrotns:getTimeSeries_WebService"
64    operation="getTimeSeries"
65    inputVariable="getTimeSeriesRequest"
66    outputVariable="getTimeSeriesResponse" />
67   <bpel:assign validate="no" name="assignReplyOutput">
68    <bpel:copy>
69     <bpel:from part="getTimeSeries_ResponsePart"
70      variable="getTimeSeriesResponse" />
71     <bpel:to part="simpleInvokeResponseTsel"
72      variable="simpleInvokeResponse" />
73    </bpel:copy>
74   </bpel:assign>
75   <bpel:reply name="replyOutput" partnerLink="simpleInvokePL"
76    portType="tns:simpleInvokePT"
77    operation="simpleInvokeOperation"
78    variable="simpleInvokeResponse" />
79  </bpel:sequence>
80
81 </bpel:process>
```

Listing B.2: WS-BPEL-Beispiel: `simpleInvoke.bpel`

C. XHydro

C.1. Übersetzung der XML Schema Entwurf Namen in XML Schema Element Namen

XML Schema Entwurf Name	XML Schema Element Name
Aggregation	ag
Area	ar
Binary	b
Category	c
data	d
deviceDescription	dd
deviceName	dn
distance	dst
dataType	dt
dataTypeCode	dtc
frequency	f
interpolation	ip
isochron	iso
interval	it
level	l
lDescription	ld
locationDescription	ldn
lName	ln
line	lne
mimetype	mt
oDescription	od
oName	on
organization	org
offset	ot
parameter	p
parameterDevice	pd
parameterMetaData	pmd

XML Schema Entwurf Name	XML Schema Element Name
parameterMetaDataList	pmdl
point	pt
point3D	pt3d
px	px
py	py
pz	pz
referenceSystem	rs
space	sp
startTimeStamp	sts
timedDataElement	tde
timeStamp	ts
timeStampDevice	tsd
timeSeries	tse
timeSeriesList	tsel
timeStampMetaData	tsmd
timeStampMeasurementInaccuracy	tsmi
timeStampPositionCode	tsp
timeStampQuality	tsq
timeStampQualityRemark	tsqr
timeStampRating	tsr
tsValue	tsv
unit	u
dataValue	v
dAnyValue	va
dBinaryValue	vb
dFloatValue	vf
dIntegerValue	vi
values	vls
dataValueMeasurementInaccuracy	vmi
dataValueQuality	vq
dataValueQualityRemark	vqr
dataValueRating	vr
dStringValue	vs
xCategory	xc
xDataTypeCode	xdtc
exchangeId	xid
exchangeIds	xids
xInterpolation	xip

XML Schema Entwurf Name	XML Schema Element Name
exchangeIdKey	xk
xParameter	xp
xReferenceSystem	xrs
xTimeStampPositionCode	xtsp
xTimeStampQualityRemark	xtsqr
xUnit	xu
exchangeIdValue	xv
xDataValueQualityRemark	xvqr

Tabelle C.1.: Übersetzung der XML Schema Entwurf Namen in XML Schema Element Namen [Kd07b]

C.2. XHydro-Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<tsel xmlns="http://xhydro.org/archive/2007/06"
 xmlns:d="http://www.disy.net/device"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xhydro.org/archive/2007/06
  http://www.xhydro.org/archive/2007/06/XHydroArchive.xsd">
 <ext />
 <tse>
  <xids>
   <xid>
    <ext />
    <xk>disy</xk>
    <xv>4711</xv>
   </xid>
  </xids>
  <org>
   <on>disy</on>
   <od>A company.</od>
  </org>
  <iso>
   <dst>P1D</dst>
   <sts>
    <tsq>
     <tsmi>1.5E-6</tsmi>
    </tsq>
    <tsv>2001-12-31T12:00:00</tsv>
   </sts>
  </iso>
  <pmdl>
```

```
    <pmd>
     <xp>W</xp>
     <c>This is a non-standard category code/remark.</c>
     <xu>m</xu>
     <tl>
      <ldn>
       <ln>Europe/Germany/Karlsruhe</ln>
       <ld>Karlsruhe, a german city.</ld>
      </ldn>
     </tl>
     <dt>
      <xdtc>aggMean</xdtc>
      <ag>
       <it>P1D</it>
       <ot>PT15M</ot>
       <f>PT1M</f>
       <xtsp>begin</xtsp>
       <l>1.5</l>
      </ag>
     </dt>
     <pd>
      <ext>
       <d:serial>ABCDEFG</d:serial>
      </ext>
      <dn>dd</dn>
      <dd>A dummy disy device.</dd>
      <dl>
       <ldn>
        <ln>Europe/Germany/Karlsruhe</ln>
        <ld>Karlsruhe, a german city.</ld>
       </ldn>
      </dl>
     </pd>
     <vq>
      <vmi>5E-3</vmi>
      <vqr>
       It is a quite imprecise device, isn't it? This quality
       remark demonstrates that free-text remarks are possible,
       too.
      </vqr>
     </vq>
    </pmd>
   </pmdl>
   <tsmd>
    <tsd>
     <dn>ddts</dn>
```

```
76      <dd>A dummy dis device to measure time.</dd>
77     </tsd>
78     <tsq>
79      <tsmi>1.5E-6</tsmi>
80      <tsqr codeList="disy1" codeListAgency="disy"
81       codeListVersion="1.0">
82       ownCode
83      </tsqr>
84     </tsq>
85    </tsmd>
86    <d>
87     <tde>
88      <!-- No timestamp is given because isochron -->
89      <vls>
90       <v>
91        <vq>
92         <vmi>6E-4</vmi>
93         <xvqr>affected</xvqr>
94        </vq>
95        <vl>
96         <pt>
97          <xrs>32632</xrs>
98          <px>5.0</px>
99          <py>6.0</py>
100        </pt>
101       </vl>
102       <vf>4.5</vf>
103      </v>
104      <v>
105       <vf>4.6</vf>
106      </v>
107      <v>
108       <vq>
109        <xvqr>missing</xvqr>
110       </vq>
111       <va xsi:nil="true" />
112      </v>
113     </vls>
114    </tde>
115   </d>
116  </tse>
117 </tsel>
```

Listing C.1: XHydro-Beispiel [Bun07a]

D. Das Pflichtenheft

D.1. Produktübersicht

Das Produkt *PegelSuite* ist ein Web Service orientiertes System zur Verarbeitung von Pegeldaten. Es umfasst das Empfangen bzw. Abfragen von Daten der einzelnen Pegel, sowie deren Verarbeitung. Dazu gehören auch das Vergleichen der Sensoren einer Messstelle, die Plausibilisierung eines Gewässerabschnitts, sowie das Versenden von Alarmen. Weiterhin existiert eine Schnittstelle zur Abfrage der Daten aus dem System, mit dessen Hilfe sich die Daten beispielsweise visualisieren lassen. Die komplette Systemübersicht ist in Abbildung D.1 dargestellt und wird in Abschnitt D.1.4 erläutert.

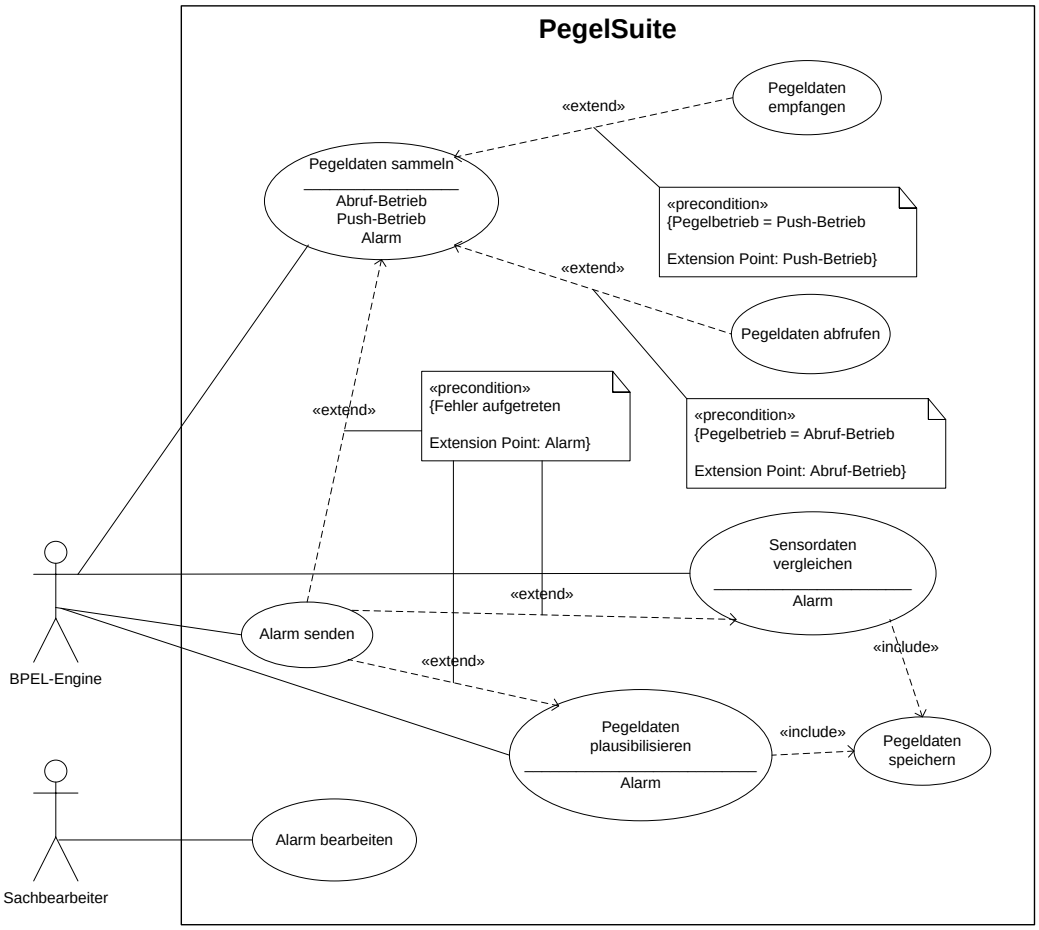


Abbildung D.1.: Systemübersicht *PegelSuite*

D.1.1. Produktziele

Das Ziel von *PegelSuite* ist die Automatisierung von Workflows zur Verarbeitung von Pegeldaten. Es soll hierfür ein standardkonformes, performantes, skalierbares System auf der Basis von Web Services geschaffen werden. Die Komponenten dieses Systems sollen austauschbar, sowie leicht wartbar sein.

D.1.2. Anwendungsbereiche

Das System findet Anwendung beim Verarbeiten der Pegeldaten von Gewässern bei der Bundesanstalt für Gewässerkunde (BfG)[1]. Es hat prototypischen Charakter und soll als Basis für ein System zur Verarbeitung von Pegeldaten dienen.

D.1.3. Zielgruppen

Die Informationen, die das System bietet, dienen der Bundesanstalt für Gewässerkunde und der Wasser- und Schifffahrtsverwaltung des Bundes zur Bewirtschaftung der Bundeswasserstraßen. Weiterhin sind die Daten für gewässernahe Einrichtungen (z.B. Campingplätze) interessant. So können beispielsweise bei Hochwasseralarm rechtzeitig Maßnahmen getroffen werden.

D.1.4. Teilprodukte

Das Produkt untergliedert sich in Teilprodukte, welche im Folgenden erläutert sind:

BPEL-Engine — Die BPEL-Engine koordiniert die Reihenfolge des Aufrufs der einzelnen Web Services.

Datenbank — Die Datenbank speichert alle Daten, welche bei der Verarbeitung der Workflows anfallen, sowie aller anderen Daten, welche für das System nötig sind.

Workflow 1: Datenabruf und -empfang — Die Pegeldaten sollen von den Messstellen sowohl abgerufen, als auch empfangen werden. Geplant ist es, dass die Messstellen die Daten selbst pushen, anstatt auf eine Abfrage zu warten. Bei nicht ankommenden Daten im Push-Betrieb soll jedoch für diese Messstelle in den Pull-Betrieb umgeschaltet werden. Generell ist der Push-Betrieb dem Pull-Betrieb vorzuziehen. Als Austauschformat wird *XHydro*[2]

[1] `http://www.bafg.de` (abgerufen: 16.07.2011)
[2] `http://www.xhydro.de` (abgerufen: 16.07.2011)

verwendet. Sollte eine Messstelle kein XHydro liefern, so werden die empfangenen Daten nach XHydro transformiert. Sollte bei nicht angekommenen Push-Daten ebenfalls das Abrufen der Daten fehlschlagen, so wird ein Alarm gesendet.

Workflow 2: Sensorvergleich — Einige Messstellen verfügen über zwei oder mehr Sensoren. Zukünftig ist es geplant alle Messstellen mit mehreren Sensoren auszustatten. Die Werte dieser Sensoren müssen verglichen und aufgrund der Unterschiede ein Qualitätsmerkmal (gleitender Mittelwert) bestimmt werden. Alle diese Werte werden dann zusammen mit dem Qualitätsmerkmal in der Datenbank gespeichert.

Zur Weiterverarbeitung soll jedoch immer der Wert des primären Sensors verwendet werden, da es sich hierbei um einen genaueren Sensor handelt. Sollte dieser einmal ausfallen, so wird in diesem Fall der Wert der sekundären Sensors verwendet. Weiterhin ist bei Ausfall eines Sensors, sowie bei einem zu großen Unterschied der Sensoren ein Alarm an die entsprechenden Stellen zu senden.

Workflow 3: Plausibilisierung — Die Rohdaten aus Workflow 2 werden, wenn genügend Daten vorhanden sind, anhand eines gesamten Gewässerabschnitts plausibilisiert. Hierzu werden die Rohdaten der Messstellen dieses Gewässerabschnitts verglichen und daraus die Qualität der Daten berechnet. Die plausibilisierten Daten werden bei erfolgreicher Plausibilisierung in der Datenbank gespeichert. Ob eine Plausibilisierung erfolgreich war, wird durch einen Vergleich der Werte mit dem Qualitätsmerkmal festgestellt.

Sollte die Plausibilisierung fehlschlagen oder nicht alle notwendigen Daten vorliegen, so soll eine Meldung an die verantwortlichen Personen gesendet werden. Diese können dann die Plausibilisierung von Hand nachtragen.

Für die Diplomarbeit ist es ausreichend, hier eine einfache Plausibilisierung zu implementieren. Hierfür kann eine Fuzzylogik der BfG verwendet werden.

Die Ausführung dieses Workflows soll stattfinden, sobald genügend Daten für eine Plausibilisierung vorliegen und ebenfalls von Hand anstoßbar sein.

Workflow 4: Alarmmanagement — Bei der Ausführung der Workflows 1, 2 und 3 kommt es immer wieder dazu, dass Alarmmeldungen an die entsprechenden Stellen gesendet werden müssen.

In folgenden Fällen ist ein Alarm vorgesehen:

- Workflow 1: Datenabruf und -empfang

  - Abruf der Daten gescheitert (Anzahl der Versuche variabel; Im Alarmzustand ankommende Daten werden weiterhin angenommen)
- Workflow 2: Sensorvergleich
  - Sensoren fehlerhaft (Im Alarmzustand ankommende Daten werden weiterhin angenommen und gespeichert)
  - Abweichung zu groß (Im Alarmzustand ankommende Daten werden weiterhin angenommen und gespeichert)
- Workflow 3: Plausibilisierung
  - Abweichung zu groß
  - Fehlende Daten

Die Alarmierung kann beispielsweise per E-Mail oder in wichtigen Fällen evtl. per SMS erfolgen. Jedoch sollen meistens nicht gleich alle zuständigen Stellen benachrichtigt werden, sondern es soll nach einer Alarmierung erst einige Zeit verstreichen, bis die nächste Stelle informiert wird. Daher ist es notwendig die Benachrichtigung zu quittieren, wenn der Fehler bearbeitet wird, so dass keine weiteren E-Mails gesendet werden. Das nicht gleichzeitige Senden der Alarmbenachrichtigungen dient dazu unklare Zuständigkeiten zu verhindern.

D.2. Produktfunktionen

- Das System kann die gepushten Daten von den Pegeln annehmen und bei Bedarf abrufen.
- Das System vergleicht die verschiedenen Sensoren einer Messstelle und legt die Messwerte zusammen mit einem Qualitätsmerkmal in der Datenbank ab.
- Das System plausibilisiert die Daten eines gesamten Gewässerabschnitts und legt diese in der Datenbank ab.
- Das System versendet bei bestimmten Ereignissen Alarme an die entsprechenden Stellen.
- Das System stellt eine Möglichkeit zur Alarmquittierung bereit.

D.3. Produktdaten

Die Daten werden durch die Pegel gepusht und können ebenfalls vom System abgerufen werden.

Die BPEL-Engine verwendet die Daten im XHydro-Format. Sollten sich ankommende Daten noch nicht in diesem Format befinden, so werden diese mit XSLT nach XHydro transformiert.

Die Datenbankmodellierung des Systems orientiert sich an der Struktur von XHydro, wobei lediglich die derzeitig verwendeten sowie für das System benötigten Elemente implementiert werden sollen.

D.4. Technische Produktumgebung

Das System setzt als Produktumgebung eine Client-Server-Architektur voraus.

D.4.1. Software

Software für Client

- Es wird ein gängiger, aktueller Browser benötigt, welcher valides HTML 4.01 interpretieren kann.
- Es wird ein Tool zur Datenbankmanipulation benötigt.

Software für Server

- Es wird ein MySQL 5.0-DMBS benötigt.
- Es wird Apache ODE 2.0 Beta 2 als BPEL-Engine benötigt
- Es wird Java SE 6 JRE benötigt.
- Es wird Apache Tomcat 7.0.0 benötigt.

D.4.2. Hardware

Hardware für Client

- Es wird ein Rechner mit Internetzugang benötigt.

Hardware für Server

- Es wird ein Rechner mit schneller Anbindung an das Internet benötigt.
- Es wird ein Rechner mit ausreichender Performance benötigt

D.4.3. Orgware

- Folgende Dokumente werden während des Projektes gepflegt:
  - Projekthandbuch
  - Projektzeitplan
  - Glossar
  - Projektbeschreibung
- Die Projektdaten unterliegen zwar nicht der Geheimhaltung, jedoch wäre eine Veröffentlichung dieser nicht wünschenswert.

D.4.4. Produkt-Schnittstellen

- Das System sollte eine XHydro-Schnittstelle für den Export und Import der Pegeldaten bieten.
- Das System sollte eine Transformations-Schnittstelle für die Transformation ankommender Daten mittels XSLT nach XHydro bieten (Beispiel: i.LON 100 des Pegels Mainz). Die XSLT-Dokumente selbst werden jedoch nicht erstellt.

D.5. Entwicklungsumgebung

Die Entwicklung des Systems wird durch Tools unterstützt. Im Folgenden werden die Anforderungen an diese festgehalten.

D.5.1. Software

- Das Modellierungswerkzeug für die Datenbank sollte im Idealfall die Konsistenzeinhaltung zwischen Datenbankschema und Datenbankimplementierung automatisiert unterstützen, d.h. dass Änderungen im Datenbankschema automatisch in die Datenbankimplementierung übernommen werden.
- Die Entwicklungsumgebung für die BPEL-Engine sollte eine einfache Orchestrierung, etwa durch grafische Modellierung, ermöglichen.

D.5.2. Hardware

- Es sind die eigenen Rechner für die Entwicklung zu benutzen. Diese sollten die Lauffähigkeit der im Projekthandbuch angegebenen Werkzeuge gewährleisten.
- Für die Datenbank werden während der Entwicklung die Universitäts-Rechner verwendet.

D.5.3. Orgware

- Die Organisation des Projektes wird durch das Projekthandbuch beschrieben.
- Es wird ein Projektzeitplan mit GanttProject gepflegt.

D.6. Anforderungen

Die folgenden Anforderungen wurden nummeriert um auf diese in anderen Dokumenten verweisen zu können. Die Nummerierung hat folgende Form:

Anforderungstyp-Nummer{**.Zwischennummer**}

Der **Anforderungstyp** wird in Kurzform angegeben. Bei **Nummer** und **Zwischennummer** handelt es sich um eine fortlaufende Nummerierung der Anforderungen. Die Angabe einer **Zwischennummer** ist nur nötig, falls neue Anforderungen, zwischen zwei Anforderungen eingefügt werden. Beim Einfügen von Anforderungen zwischen zwei Anforderungen, welche bereits eine **Zwischennummer** besitzen, wird eine weitere **Zwischennummer** angehängt. Diese Art der Nummerierung ist nötig, um die Nummerierung der bereits vorhandenen Anforderungen unverändert zu lassen. Anforderungen, welche nicht weiter gültig sind, werden durchgestrichen.

Die Anforderungen unterliegen verschiedenen Verbindlichkeiten. Diese lassen sich anhand einiger *hervorgehobener* Schlagworte (*muss (nicht)* > *soll (nicht)* > *sollte (nicht)* > *kann (nicht)*) in den Anforderungen bestimmen.

Für die funktionalen Anforderungen (siehe Abschnitt D.6.1) werden innerhalb einer Verbindlichkeitsstufe die Anforderungen durch verschiedene Prioritäten (0 (unwichtig) - 5 (wichtig)) nochmals unterteilt. Die Priorität wird *hervorgehoben* und in Klammern hinter der Anforderung genannt.

D.6.1. Funktionale Anforderungen

Workflow 1: Datenabruf und -empfang

- **F-1** Die Daten *müssen* von den Pegeln abgerufen werden können. *(Priorität: 5)*
- **F-2** Die Daten *müssen* im Push-Betrieb empfangen werden können. *(Priorität: 5)*
- **F-3** Der Push-Betrieb *muss* dem Pull-Betrieb vorgezogen werden. *(Priorität: 5)*
- **F-4** Das System *muss* nicht ankommende Daten im Push-Betrieb, im Pull-Betrieb abfragen. *(Priorität: 5)*
- **F-5** Vor dem Wechsel in den Pull-Betrieb *muss* eine variable Zeit (Standard: 15 Minuten) auf die fehlenden Daten gewartet werden. *(Priorität: 4)*
- **F-6** Es *sollten* die Daten von den Pegeln Mainz, Bingen, Kaub und Oestrich, welche jeweils die Daten von PEGELONLINE[3] durch einen Proxy abrufen und die Daten im XHydro-Format pushen, verwendet werden. *(Priorität: 5)*
- **F-7** Daten, welche in einem anderen Format als XHydro vorliegen, *sollen* durch XSLT nach XHydro transformiert werden können. *(Priorität: 3)*

[3] `http://www.pegelonline.wsv.de` (abgerufen: 16.07.2011)

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 2: Sensorvergleich

F-8 Falls Daten mehrerer Sensoren vorliegen, *müssen* diese verglichen werden. *(Priorität: 5)*

F-9 Zu den Daten *muss* ein Qualitätsmerkmal (siehe Abschnitt D.7.1) erstellt werden. *(Priorität: 5)*

F-10 Die Daten aller Sensoren *müssen* in der Datenbank gespeichert werden. *(Priorität: 5)*

F-11 Das Qualitätsmerkmal *muss* in der Datenbank gespeichert werden. *(Priorität: 5)*

F-12 Wenn nur ein Sensor existiert, so *soll* das Qualitätsmerkmal `NULL` sein. *(Priorität: 4)*

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 3: Plausibilisierung

F-13 Die Daten eines gesamten Gewässerabschnitts *müssen* plausibilisiert werden (siehe Abschnitt D.7.2). *(Priorität: 5)*

F-14 ~~Die Qualität der plausibilisierten Daten (gleitender Mittelwert; temporär) *muss* berechnet werden. *(Priorität: 5)*~~

F-15 Zur Plausibilisierung *muss* der Pegelwert des höchst priorisierten, funktionierenden Sensors genommen werden. *(Priorität: 5)*

F-16 Die plausibilisierten Daten *müssen* in der Datenbank gespeichert werden. *(Priorität: 5)*

F-16.1 Es *müssen* nur die Daten des an der Plausibilisierung beteiligten Sensors als plausibilisierte Daten übernommen werden. *(Priorität: 5)*

F-17 ~~Der Workflow *soll* automatisch nach einer bestimmten Zeitspanne (variabel) gestartet werden. *(Priorität: 3)*~~

F-17.1 Der Workflow *soll* von Hand aufrufbar sein. *(Priorität: 4)*

F-17.2 Der Workflow *soll* gestartet werden, sobald genügend Daten für die Plausibilisierung vorliegen. *(Priorität: 3)*

Weitere funktionale Anforderungen, die diesen Workflow bezüglich des Alarmmanagements betreffen, werden in dem Abschnitt „Workflow 4: Alarmmanagement“ beschrieben.

Workflow 4: Alarmmanagement

F-18 Das System *muss* Alarme per E-Mail versenden können. *(Priorität: 5)*

F-19 Das System *kann* in wichtigen Fällen Alarme per SMS verschicken. *(Priorität: 0)*

F-20 ~~Es *sollen nicht* alle zuständigen Stellen gleichzeitig benachrichtigt werden. *(Priorität: 4)*~~

F-20.1 Es *sollen* alle Personen innerhalb einer Eskalationsstufe gleichzeitig benachrichtigt werden. *(Priorität: 4)*

F-20.2 Eine Person *soll* im Alarmfall über alle angegebenen Kontaktadressen gleichzeitig benachrichtigt werden. *(Priorität: 3)*

F-21 Es *soll* erst nach Ablauf einer variablen Zeit die nächste Stelle benachrichtigt werden. *(Priorität: 4)*

F-22 Die Zeit bis zur nächsten Eskalationsstufe *kann* pro Person einstellbar sein. *(Priorität: 0)*

F-22.1 Vor dem Wechsel in die nächste Eskalationsstufe *kann* der Alarm, beim Ausbleiben von Quittierungen, mehrmals wiederholt werden. *(Priorität: 0)*

F-23 Die Bearbeitung eines Alarms *muss* quittiert werden. *(Priorität: 3)*

F-23.1 Die Zuständigkeit für einen Alarm *muss* pro Pegel und Alarmtyp konfigurierbar sein. *(Priorität: 3)*

- Es *muss* ein Alarm gesendet werden, ...
  - Workflow 1: Datenabruf und -empfang

    F-24 ...wenn der Abruf der Daten gescheitert ist. *(Priorität: 5)*

    F-25 Die Anzahl der Versuche, bevor ein Alarm gesendet wird *soll* variabel (Standard: 1 mal) sein. *(Priorität: 3)*

    F-26 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen werden. *(Priorität: 3)*
  - Workflow 2: Sensorvergleich

    F-27 ...wenn Sensoren fehlerhaft sind. *(Priorität: 5)*

    F-28 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen und gespeichert werden. *(Priorität: 3)*

    F-29 ...wenn die Abweichung der Daten zum gleitenden Mittelwert zu groß ist. *(Priorität: 4)*

    F-30 Im Alarmzustand ankommende Daten *sollen* weiterhin angenommen und gespeichert werden. *(Priorität: 2)*

    F-31 Die erlaubte Abweichung *soll* variabel (Standard: 2 - 3 cm) sein. *(Priorität: 2)*

    F-32 ~~Die automatische Ausführung von „Workflow 3: Plausibilisierung“ *muss* gestoppt werden. *(Priorität: 4)*~~
  - Workflow 3: Plausibilisierung

F-33 ~~. . . wenn die Abweichung der plausibilisierten Daten zum gleitenden Mittelwert zu groß ist. *(Priorität: 4)*~~

F-34 ~~Die erlaubte Abweichung *soll* variabel (Standard: 2 - 3 cm) sein. *(Priorität: 2)*~~

F-35 ~~Die automatische Ausführung von „Workflow 3: Plausibilisierung“ *muss* gestoppt werden. *(Priorität: 4)*~~

F-35.1 ~~Die derzeitige (für diesen Zeitraum und Gewässerabschnitt) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*~~

F-35.1.1 . . . wenn die Daten nicht plausibilisiert werden konnten. *(Priorität: 4)*

F-35.1.2 Die derzeitige (für diesen Zeitraum und Pegel) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*

F-35.2 Abgebrochene Plausibilisierungen *müssen* von Hand nachgetragen oder „Workflow 3: Plausibilisierung“ manuell gestartet werden. *(Priorität: 4)*

F-36 . . . wenn zur Plausibilisierung benötigte Daten fehlen. *(Priorität: 5)*

F-37 ~~Die automatische Ausführung von „Workflow 3: Plausibilisierung“~~
~~*muss* gestoppt werden. *(Priorität: 4)*~~

F-37.1 ~~Die derzeitige (für diesen Zeitraum und Gewässerabschnitt) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*~~

F-37.1.1 Die derzeitige (für diesen Zeitraum und Pegel) Ausführung von „Workflow 3: Plausibilisierung“ *muss* abgebrochen werden. *(Priorität: 4)*

F-37.2 Abgebrochene Plausibilisierungen *müssen* von Hand nachgetragen oder „Workflow 3: Plausibilisierung“ manuell gestartet werden. *(Priorität: 4)*

Authentifizierung und Integritätsprüfung

F-38 Es *sollte* eine Authentifizierung und Integritätsprüfung zwischen den Pegeln und der BPEL-Engine stattfinden. *(Priorität: 1)*

F-39 Die Authentifizierung und Integritätsprüfung *sollte* optional zuschaltbar sein. *(Priorität: 1)*

D.6.2. Technische Anforderungen

T-1 Das System *muss* auf Web Services basieren.

T-1.1 Das System *muss* frei verfügbare Software verwenden.

T-2 Das System *soll* SOAP über HTTP verwenden.
T-3 Die Web Services *sollen* in Java implementiert werden.
T-4 Die Web Services *müssen* von einer BPEL-Engine orchestriert werden.
T-5 Als BPEL-Engine *sollte* Apache ODE verwendet werden.
T-6 Zur Verarbeitung von XML innerhalb von Java *soll* die Streaming API for XML (StAX) verwendet werden.
T-7 Das Web-Interface *soll* mit JSF implementiert werden.
T-8 Das System *muss* im unbeaufsichtigten Betrieb laufen.
T-9 Das System *soll* rund um die Uhr erreichbar sein.
T-10 Das System *muss* sich an bestehende Standards halten.
T-11 Die einzelnen Komponenten des Systems *sollen* leicht austauschbar sein.

Datenbank

T-12 Das Datenbanksystem *soll* MySQL sein.
T-13 Die Datenbank *soll* als Storage-Engine InnoDB verwenden.
T-14 Das Datenbank Encoding *muss* UTF-8 sein.
T-14.1 Auf die Datenbank soll über eine XHydro-Schnittstelle (ausgehend und eingehend) zugegriffen werden.

XHydro

T-15 Innerhalb des Systems *soll* XHydro als Datenaustauschformat verwendet werden.
T-16 Die Verschlüsselung für XHydro *soll nicht* verwendet werden.
T-17 Das System *soll* die Signatur nach XML-Signature für XHydro verwenden.
T-18 Für die Kompression von XHydro *soll* die HTTP-Komprimierung verwendet werden, sofern diese von beiden Kommunikationspartnern unterstützt wird.

Authentifizierung und Integritätsprüfung

T-19 Es *sollte* der Standard WS-Security verwendet werden.
T-20 Die Realisierung der Authentifizierung und Integritätsprüfung *sollte* auf einer Schicht unter der Anwendungsschicht der OSI-Modells stattfinden.

D.6.3. Qualitätsanforderungen

Die folgenden Qualitätsanforderungen orientieren sich am Industriestandard. Diese wird als Standardqualität definiert.

#	Qualität	sehr gut	gut	normal
	Funktionalität			
Q-1	Angemessenheit		✘	
Q-2	Richtigkeit		✘	
Q-3	Interoperabilität	✘		
Q-4	Ordnungsmäßigkeit		✘	
Q-5	Sicherheit			✘
	Zuverlässigkeit			
Q-6	Reife			✘
Q-7	Fehlertoleranz			✘
Q-8	Wiederherstellbarkeit			✘
	Benutzbarkeit			
Q-9	Verständlichkeit	✘		
Q-10	Erlernbarkeit	✘		
Q-11	Bedienbarkeit	✘		
	Effizienz			
Q-12	Zeitverhalten	✘		
Q-13	Verbrauchsverhalten			✘
	Änderbarkeit			
Q-14	Analysierbarkeit		✘	
Q-15	Modifizierbarkeit	✘		
Q-16	Stabilität			✘
Q-17	Prüfbarkeit			✘
	Übertragbarkeit			
Q-18	Anpassbarkeit		✘	
Q-19	Installierbarkeit			✘
Q-20	Konformität	✘		
Q-21	Austauschbarkeit	✘		

Tabelle D.1.: Qualitätsanforderungen

D.6.4. Anforderungen an begleitende Dokumente

D-1 Die begleitenden Dokumente *müssen* während des gesamten Entwicklungsprozess gepflegt werden.

D-2 Dokumente *sollten* abgeschlossen werden (Text „Finale Version“ unter Titel), wenn die Pflege nicht mehr sinnvoll erscheint.

D.6.5. Anforderungen an den Entwicklungsprozess

E-1 Das zu modellierende Datenbankschema *muss* sich an XHydro orientieren.

E-2 Es *soll* eine datenbankunabhängige Java-Schnittstelle (etwa JDBC) verwendet werden.

D.6.6. Rechtlich-vertragliche Anforderungen

Da es sich bei dem Projekt um eine Diplomarbeit handelt, existieren keine rechtlich-vertraglichen Anforderungen.

D.7. Ergänzungen

D.7.1. Workflow 2: Qualitätsmerkmal

Bei dem Qualitätsmerkmal handelt es sich um die Abweichung des gleitenden Mittelwertes. Er entspricht dem XHydro-Element `measurementInaccuracy`.

D.7.2. Workflow 3: Plausibilisierung

Die Plausibilisierung wird durch eine Fuzzylogik der Bundesanstalt für Gewässerkunde ausgeführt. Aus den daraus resultierenden Werten wird der gleitende Mittelwert gebildet und dieser mit den Daten vergleichen. Anhand der Abweichung lässt sich bestimmen, ob die Plausibilisierung erfolgreich war. Die Qualität wird nicht in der Datenbank gespeichert, sondern ist lediglich temporär.

D.7.3. Weitere Informationen

Weitere Informationen befinden sich u.a. im Projekthandbuch.

E. Der Datenbankentwurf

E.1. PegelSuite SQL-Schema

```
-- ---------------------------------------- --
-- This sql file contains the tables and     --
-- constraints of the PegelSuite database.   --
-- ---------------------------------------- --

-- ------------- --
-- Create Tables --
-- ------------- --

-- This table holds information about the gauges
CREATE TABLE Gauge
(
  gaugeID INT NOT NULL AUTO_INCREMENT,
  locationNumber TEXT NOT NULL,
  locationName TEXT NOT NULL,
  ipAddress TEXT NOT NULL,
  getTimeSeriesPath TEXT NOT NULL,
  exchangeFormatID INT NOT NULL,
  waterID INT NOT NULL,
  pnp DOUBLE,
  riverKilometer DOUBLE,
  wsaID INT NOT NULL,
  equidistanceOfPushes INT NOT NULL,
  toleranceOfPushes INT NOT NULL,
  equidistanceOfMeasurements INT NOT NULL,
  maxPullRetries INT NOT NULL,
  maxSensorDeviation DOUBLE NOT NULL,
  maxValidationDeviation DOUBLE NOT NULL,
  isActive BOOLEAN NOT NULL,
  PRIMARY KEY (gaugeID)
) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;

-- This table holds information about the waters
CREATE TABLE Water
(
```

```
36    waterID INT NOT NULL AUTO_INCREMENT,
37    name TEXT,
38    waterAreaID INT NOT NULL,
39    PRIMARY KEY (waterID)
40  ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
41
42  -- This table holds information about the water areas
43  CREATE TABLE WaterArea
44  (
45    waterAreaID INT NOT NULL AUTO_INCREMENT,
46    name TEXT,
47    PRIMARY KEY (waterAreaID)
48  ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
49
50  -- This table holds information about the order of the gauges
51  CREATE TABLE GaugeIsFollowedByGauge
52  (
53    gaugeID INT NOT NULL,
54    followingGaugeID INT NOT NULL,
55    PRIMARY KEY (gaugeID, followingGaugeID)
56  ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
57
58  -- This table holds information about the geo references
59  -- of the gauges
60  CREATE TABLE GeoReference
61  (
62    geoReferenceID INT NOT NULL AUTO_INCREMENT,
63    gaugeID INT NOT NULL,
64    x DOUBLE,
65    y DOUBLE,
66    z DOUBLE,
67    geoReferenceSystemID INT NOT NULL,
68    PRIMARY KEY (geoReferenceID)
69  ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
70
71  -- This table holds information about the Waterways and Shipping
72  -- Offices (WSA)
73  CREATE TABLE WSA
74  (
75    wsaID INT NOT NULL AUTO_INCREMENT,
76    name TEXT,
77    description TEXT,
78    wsdID INT NOT NULL,
79    PRIMARY KEY (wsaID)
80  ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
81
```

```
82 -- This table holds information about the Waterways and Shipping
83 -- Directorates (WSD)
84 CREATE TABLE WSD
85 (
86   wsdID INT NOT NULL AUTO_INCREMENT,
87   name TEXT,
88   description TEXT,
89   PRIMARY KEY (wsdID)
90 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
91
92 -- This table holds information about the sensors
93 CREATE TABLE Sensor
94 (
95   sensorID INT NOT NULL AUTO_INCREMENT,
96   gaugeID INT NOT NULL,
97   sensorNumber INT NOT NULL,
98   parameterID INT NOT NULL,
99   isActive BOOLEAN NOT NULL,
100  PRIMARY KEY (sensorID)
101 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
102
103 -- This table holds information about the parameters
104 CREATE TABLE Parameter
105 (
106   parameterID INT NOT NULL AUTO_INCREMENT,
107   name TEXT NOT NULL,
108   parameterCodeID INT,
109   unitID INT,
110   PRIMARY KEY (parameterID)
111 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
112
113 -- This table holds information about the parameter codes
114 CREATE TABLE ParameterCode
115 (
116   parameterCodeID INT NOT NULL AUTO_INCREMENT,
117   parameterCode TEXT NOT NULL,
118   description TEXT,
119   PRIMARY KEY (parameterCodeID)
120 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
121
122 -- This table holds information about the units
123 CREATE TABLE Unit
124 (
125   unitID INT NOT NULL AUTO_INCREMENT,
126   unit TEXT NOT NULL,
127   description TEXT,
```

```
128   PRIMARY KEY (unitID)
129 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
130
131 -- This table holds information about the notification templates
132 CREATE TABLE Priority
133 (
134   priorityID INT NOT NULL AUTO_INCREMENT,
135   priority TEXT NOT NULL,
136   description TEXT,
137   PRIMARY KEY (priorityID)
138 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
139
140 -- This table holds the enumeration of the alert types
141 CREATE TABLE AlertType
142 (
143   alertTypeID INT NOT NULL AUTO_INCREMENT,
144   alertType TEXT NOT NULL,
145   description TEXT,
146   PRIMARY KEY (alertTypeID)
147 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
148
149 -- This table holds the enumeration of the contact types
150 CREATE TABLE ContactType
151 (
152   contactTypeID INT NOT NULL AUTO_INCREMENT,
153   contactType TEXT NOT NULL,
154   description TEXT,
155   PRIMARY KEY (contactTypeID)
156 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
157
158 -- This table holds the enumeration of the data type codes
159 CREATE TABLE DataTypeCode
160 (
161   dataTypeCodeID INT NOT NULL AUTO_INCREMENT,
162   dataTypeCode TEXT NOT NULL,
163   description TEXT,
164   PRIMARY KEY (dataTypeCodeID)
165 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
166
167 -- This table holds the enumeration of the exchange formats
168 CREATE TABLE ExchangeFormat
169 (
170   exchangeFormatID INT NOT NULL AUTO_INCREMENT,
171   exchangeFormat TEXT NOT NULL,
172   description TEXT,
173   PRIMARY KEY (exchangeFormatID)
```

```
174 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
175
176 -- This table holds the enumeration of the alert states
177 CREATE TABLE AlertState
178 (
179   alertStateID INT NOT NULL AUTO_INCREMENT,
180   alertState TEXT NOT NULL,
181   description TEXT,
182   PRIMARY KEY (alertStateID)
183 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
184
185 -- This table holds the enumeration of the geo reference systems
186 CREATE TABLE GeoReferenceSystem
187 (
188   geoReferenceSystemID INT NOT NULL AUTO_INCREMENT,
189   geoReferenceSystem TEXT NOT NULL,
190   description TEXT,
191   PRIMARY KEY (geoReferenceSystemID)
192 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
193
194 -- This table holds the information of the raw data
195 CREATE TABLE RawData
196 (
197   sensorID INT NOT NULL,
198   timeStampOfMeasurements DATETIME NOT NULL,
199   measuredValue DOUBLE NOT NULL,
200   dataTypeCodeID INT,
201   qualityRemark DOUBLE,
202   isValidated BOOLEAN,
203   PRIMARY KEY (sensorID, timeStampOfMeasurements)
204 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
205
206 -- This table holds the information of the validated data
207 CREATE TABLE ValidatedData
208 (
209   sensorID INT NOT NULL,
210   timeStampOfMeasurements DATETIME NOT NULL,
211   measuredValue DOUBLE NOT NULL,
212   dataTypeCodeID INT,
213   qualityRemark DOUBLE,
214   PRIMARY KEY (sensorID, timeStampOfMeasurements)
215 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
216
217 -- This table holds information about the alerts
218 CREATE TABLE Alert
219 (
```

```
220   alertID INT NOT NULL AUTO_INCREMENT,
221   alertTypeID INT NOT NULL,
222   alertStateID INT NOT NULL,
223   gaugeID INT NOT NULL,
224   affectedSensorID INT,
225   alertTimeStamp DATETIME,
226   escalationLevel INT,
227   notificationRetries INT,
228   lastNotificationTimeStamp DATETIME,
229   PRIMARY KEY (alertID)
230 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
231
232 -- This table holds information about the notification templates
233 CREATE TABLE NotificationTemplate
234 (
235   alertTypeID INT NOT NULL,
236   alertStateID INT NOT NULL,
237   contactTypeID INT NOT NULL,
238   priorityID INT NOT NULL,
239   subject TEXT NOT NULL,
240   body TEXT NOT NULL,
241   PRIMARY KEY (alertTypeID, alertStateID, contactTypeID)
242 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
243
244 -- This table holds information about the missing data
245 CREATE TABLE MissingData
246 (
247   sensorID INT NOT NULL,
248   latestDataTimeStamp DATETIME NOT NULL,
249   fromTimeStamp DATETIME NOT NULL,
250   toTimeStamp DATETIME NOT NULL,
251   pullRetries INT NOT NULL,
252   PRIMARY KEY (sensorID, latestDataTimeStamp)
253 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
254
255 -- This table holds information about the last receipt data
256 CREATE TABLE ReceiptInformation
257 (
258   sensorID INT NOT NULL,
259   lastPushTimeStamp DATETIME NOT NULL,
260   latestDataTimeStamp DATETIME NOT NULL,
261   PRIMARY KEY (sensorID)
262 ) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;
263
264 -- This table holds information about the person
265 CREATE TABLE Person
```

```
(
  personID INT NOT NULL AUTO_INCREMENT,
  firstName TEXT NOT NULL,
  lastName TEXT NOT NULL,
  maxNotificationRetries INT NOT NULL,
  equidistanceOfNotifications INT NOT NULL,
  PRIMARY KEY (personID)
) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;

-- This table holds information about the contact data of a
-- person
CREATE TABLE Contact
(
  contactID INT NOT NULL AUTO_INCREMENT,
  personID INT NOT NULL,
  address TEXT NOT NULL,
  contactTypeID INT NOT NULL,
  PRIMARY KEY (contactID)
) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;

-- This table holds information about the responsibility of a
-- person
CREATE TABLE Responsibility
(
  personID INT NOT NULL,
  gaugeID INT NOT NULL,
  alertTypeID INT NOT NULL,
  escalationLevel INT NOT NULL,
  PRIMARY KEY (personID, gaugeID, alertTypeID)
) ENGINE = InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_unicode_ci;

-- --------------- --
-- Add Constraints --
-- --------------- --

ALTER TABLE Gauge ADD CONSTRAINT GaugeHasExchangeFormat
  FOREIGN KEY (exchangeFormatID)
   REFERENCES ExchangeFormat(exchangeFormatID)
  ON DELETE RESTRICT ON UPDATE CASCADE
;

ALTER TABLE Gauge ADD CONSTRAINT GaugeIsPartOfWater
  FOREIGN KEY (waterID) REFERENCES Water(waterID)
  ON DELETE RESTRICT ON UPDATE CASCADE
;

```

```
312 ALTER TABLE Gauge ADD CONSTRAINT GaugeIsOperatedByWSA
313   FOREIGN KEY (wsaID) REFERENCES WSA(wsaID)
314   ON DELETE RESTRICT ON UPDATE CASCADE
315 ;
316
317 ALTER TABLE Water ADD CONSTRAINT WaterIsPartOfWaterArea
318   FOREIGN KEY (waterAreaID) REFERENCES WaterArea(waterAreaID)
319   ON DELETE RESTRICT ON UPDATE CASCADE
320 ;
321
322 ALTER TABLE GaugeIsFollowedByGauge ADD CONSTRAINT ActualGauge
323   FOREIGN KEY (gaugeID) REFERENCES Gauge(gaugeID)
324   ON DELETE RESTRICT ON UPDATE CASCADE
325 ;
326
327 ALTER TABLE GaugeIsFollowedByGauge ADD CONSTRAINT FollowingGauge
328   FOREIGN KEY (followingGaugeID) REFERENCES Gauge(gaugeID)
329   ON DELETE RESTRICT ON UPDATE CASCADE
330 ;
331
332 ALTER TABLE GeoReference
333  ADD CONSTRAINT GeoReferenceBelongsToGauge
334   FOREIGN KEY (gaugeID) REFERENCES Gauge(gaugeID)
335   ON DELETE RESTRICT ON UPDATE CASCADE
336 ;
337
338 ALTER TABLE GeoReference
339  ADD CONSTRAINT GeoReferenceHasGeoReferenceSystem
340   FOREIGN KEY (geoReferenceSystemID)
341    REFERENCES GeoReferenceSystem(geoReferenceSystemID)
342   ON DELETE RESTRICT ON UPDATE CASCADE
343 ;
344
345 ALTER TABLE WSA ADD CONSTRAINT WSAIsUnderControlOfWSD
346   FOREIGN KEY (wsdId) REFERENCES WSD(wsdID)
347   ON DELETE RESTRICT ON UPDATE CASCADE
348 ;
349
350 ALTER TABLE Sensor ADD CONSTRAINT SensorIsPartOfGauge
351   FOREIGN KEY (gaugeID) REFERENCES Gauge(gaugeID)
352   ON DELETE RESTRICT ON UPDATE CASCADE
353 ;
354
355 ALTER TABLE Sensor ADD CONSTRAINT SensorMeasuresParameter
356   FOREIGN KEY (parameterID) REFERENCES Parameter(parameterID)
357   ON DELETE RESTRICT ON UPDATE CASCADE
```

```
358 ;
359
360 ALTER TABLE Parameter ADD CONSTRAINT ParamaterHasParameterCode
361   FOREIGN KEY (parameterCodeID)
362    REFERENCES ParameterCode(parameterCodeID)
363   ON DELETE RESTRICT ON UPDATE CASCADE
364 ;
365
366 ALTER TABLE Parameter ADD CONSTRAINT ParamaterHasUnit
367   FOREIGN KEY (unitID) REFERENCES Unit(unitID)
368   ON DELETE RESTRICT ON UPDATE CASCADE
369 ;
370
371 ALTER TABLE RawData ADD CONSTRAINT RawDataHasDataTypeCode
372   FOREIGN KEY (dataTypeCodeID)
373    REFERENCES DataTypeCode(dataTypeCodeID)
374   ON DELETE SET NULL ON UPDATE CASCADE
375 ;
376
377 ALTER TABLE RawData ADD CONSTRAINT RawDataBelongsToSensor
378   FOREIGN KEY (sensorID) REFERENCES Sensor(sensorID)
379   ON DELETE RESTRICT ON UPDATE CASCADE
380 ;
381
382 ALTER TABLE ValidatedData
383  ADD CONSTRAINT ValidatedDataHasDataTypeCode
384   FOREIGN KEY (dataTypeCodeID)
385    REFERENCES DataTypeCode(dataTypeCodeID)
386   ON DELETE SET NULL ON UPDATE CASCADE
387 ;
388
389 ALTER TABLE ValidatedData
390  ADD CONSTRAINT ValidatedDataBelongsToSensor
391   FOREIGN KEY (sensorID) REFERENCES Sensor(sensorID)
392   ON DELETE RESTRICT ON UPDATE CASCADE
393 ;
394
395 ALTER TABLE Alert ADD CONSTRAINT AlertHasAlertType
396   FOREIGN KEY (alertTypeID) REFERENCES AlertType(alertTypeID)
397   ON DELETE RESTRICT ON UPDATE CASCADE
398 ;
399
400 ALTER TABLE Alert ADD CONSTRAINT AlertHasAffectedSensor
401   FOREIGN KEY (affectedSensorID) REFERENCES Sensor(sensorID)
402   ON DELETE SET NULL ON UPDATE CASCADE
403 ;
```

```
404
405 ALTER TABLE Alert ADD CONSTRAINT AlertHasAlertState
406   FOREIGN KEY (alertStateID) REFERENCES AlertState(alertStateID)
407   ON DELETE RESTRICT ON UPDATE CASCADE
408 ;
409
410 ALTER TABLE Alert ADD CONSTRAINT AlertBelongsToGauge
411   FOREIGN KEY (gaugeID) REFERENCES Gauge(gaugeID)
412   ON DELETE RESTRICT ON UPDATE CASCADE
413 ;
414
415 ALTER TABLE NotificationTemplate
416  ADD CONSTRAINT NotificationTemplateHasAlertType
417   FOREIGN KEY (alertTypeID) REFERENCES AlertType(alertTypeID)
418   ON DELETE CASCADE ON UPDATE CASCADE
419 ;
420
421 ALTER TABLE NotificationTemplate
422  ADD CONSTRAINT NotificationTemplateHasAlertState
423   FOREIGN KEY (alertStateID) REFERENCES AlertState(alertStateID)
424   ON DELETE CASCADE ON UPDATE CASCADE
425 ;
426
427 ALTER TABLE NotificationTemplate
428  ADD CONSTRAINT NotificationTemplateHasContactType
429   FOREIGN KEY (contactTypeID)
430    REFERENCES ContactType(contactTypeID)
431   ON DELETE CASCADE ON UPDATE CASCADE
432 ;
433
434 ALTER TABLE NotificationTemplate
435  ADD CONSTRAINT NotificationTemplateHasPriority
436   FOREIGN KEY (priorityID) REFERENCES Priority(priorityID)
437   ON DELETE RESTRICT ON UPDATE CASCADE
438 ;
439
440 ALTER TABLE MissingData ADD CONSTRAINT MissingDataBelongsToSensor
441   FOREIGN KEY (sensorID) REFERENCES Sensor(sensorID)
442   ON DELETE RESTRICT ON UPDATE CASCADE
443 ;
444
445 ALTER TABLE ReceiptInformation
446  ADD CONSTRAINT ReceiptInformationBelongsToSensor
447   FOREIGN KEY (sensorID) REFERENCES Sensor(sensorID)
448   ON DELETE RESTRICT ON UPDATE CASCADE
449 ;
```

```
450
451 ALTER TABLE Contact ADD CONSTRAINT ContactHasContactType
452   FOREIGN KEY (contactTypeID)
453    REFERENCES ContactType(contactTypeID)
454   ON DELETE RESTRICT ON UPDATE CASCADE
455 ;
456
457 ALTER TABLE Contact ADD CONSTRAINT ContactBelongsToPerson
458   FOREIGN KEY (personID) REFERENCES Person(personID)
459   ON DELETE CASCADE ON UPDATE CASCADE
460 ;
461
462 ALTER TABLE Responsibility ADD CONSTRAINT ResponsiblePerson
463   FOREIGN KEY (personID) REFERENCES Person(personID)
464   ON DELETE RESTRICT ON UPDATE CASCADE
465 ;
466
467 ALTER TABLE Responsibility ADD CONSTRAINT GaugeOfResponsibility
468   FOREIGN KEY (gaugeID) REFERENCES Gauge(gaugeID)
469   ON DELETE RESTRICT ON UPDATE CASCADE
470 ;
471
472 ALTER TABLE Responsibility
473  ADD CONSTRAINT AlertTypeOfResponsibility
474   FOREIGN KEY (alertTypeID) REFERENCES AlertType(alertTypeID)
475   ON DELETE RESTRICT ON UPDATE CASCADE
476 ;
```

Listing E.1: PegelSuite SQL-Schema

F. Glossar

Begriff	Erklärung
BPEL	➤ Web Services Business Process Execution Language
BPEL-Engine	Eine *BPEL-Engine* dient als Laufzeitumgebung für die BPEL-Prozesse und ist für die Interpretation und Ausführung der BPEL-Datei zuständig [Bra09].
Endpunkt	Ein *Endpunkt* beschreibt eine spezifische Adresse für den Zugriff auf einen Service. Dies geschieht durch die Angabe eines spezifischen Protokolls und Datenformats. [W3C04b]
Erweiterungselement (WSDL)	Ein *Erweiterungselement* (WSDL) bezeichnet ein Element, innerhalb einer WSDL-Datei, welches eine spezifische Technologie (z.B. SOAP) repräsentiert. Es kann an den dafür vorgesehenen Stellen eingefügt werden und muss einen anderen als den WSDL-Namensraum besitzen. [W3C01]
Eskalationsstufe	Im Rahmen der Diplomarbeit bezeichnet eine *Eskalationsstufe* den Stand der Alarmierung. Jedes Benachrichtigungsintervall entspricht dabei einer Eskalationsstufe.
Extensible Markup Language	Die *Extensible Markup Language* (XML) ist eine *Auszeichnungssprache*, welche *hierarchisch strukturierte Daten* und teilweise auch *das Verhalten der verarbeitenden Programme beschreibt*. [W3C08a]

Begriff	Erklärung
Fuzzylogik	*Fuzzylogik* ist eine Theorie für die Modellierung von Unsicherheiten und Unschärfen insbesondere bei technischen Problemen [Ste03]. Im Rahmen der Diplomarbeit bezeichnet die Fuzzylogik auch das Programm zur Plausibilisierung eines Gewässerabschnitts.
Geschäftsprozess	➤ Workflow
Hypermedia	*Hypermedia* bezeichnet eine Schnittmenge von ➤ Hypertext und ➤ Multimedia [Pro03]. Die Struktur von Hypermedia besteht aus Knoten und Verbindungen zwischen diesen. Die Knoten repräsentieren Informationen in digitalisierter Form. Operationen (z.B. das Anlegen oder Lesen) auf Hypermedia sind im Allgemeinen nicht sequentiell und werden interaktiv, im Allgemeinen über eine Benutzeroberfläche, ausgeführt. Die Inhalte von Hypermedia werden multimodal (mehrere Sinne ansprechend) und multicodal (mehrere Zeichensysteme; z.B. Bild und Text) präsentiert. [Blu98]
Hypertext	Die Struktur von *Hypertext* besteht aus Knoten und Verbindungen zwischen diesen. Die Knoten repräsentieren Informationen. Operationen (z.B. das Anlegen oder Lesen) auf Hypertexten sind im Allgemeinen nicht sequentiell und werden interaktiv, im Allgemeinen über eine Benutzeroberfläche ausgeführt. Die Inhalte von Hypertexten sind durch statische Medien textuell, bildhaft oder symbolisch repräsentiert. [Blu98]
isochron	Zeiträume mit äquidistantem Abstand werden *isochron* genannt.
Komponente (REST)	Eine *Komponente* ist eine Software-Einheit, welche zusammen mit ihrem internen Zustand die Transformation von Daten über ein Interface bereitstellt. [Fie00]

Begriff	Erklärung
Konnektor (REST)	Bei einem *Konnektor* handelt es sich um einen abstrakten Mechanismus, welcher die Kommunikation, Koordination und Kooperation zwischen Komponenten regelt. [Fie00]
M2M	➤ Machine-to-Machine
Machine-to-Machine	*Machine-to-Machine*, bezeichnet den automatischen Datenaustausch zwischen Maschinen. [Wal07]
Multimedia	Als *Multimedia* bezeichnet man digitalsierte, integrierte, multimodal (mehrere Sinne ansprechend) und multicodal (mehrere Zeichensysteme; z.B. Bild und Text) präsentierte Informationen. Diese können von einem Anwender interaktiv genutzt werden. [Blu98]
Namensraum (XML)	*XML-Namensräume* dienen der Unterscheidung verschiedener XML-Sprachen innerhalb eines Dokuments. Mit ihnen lassen sich etwa Tags unterscheiden, die in den verschiedenen XML-Sprachen gleich benannt sind. [W3C09a]
Orchestrierung	Die Zusammenstellung mehrerer Web Services zu einem Geschäftsprozess wird als *Orchestrierung* bezeichnet. [Vas07]
Pegel	Ein *Pegel* ist ein Meßgerät zu Bestimmung des Wasserstandes. Im weiteren Sinne bezeichnet es die Messstelle selbst.
Pegeldaten	Als *Pegeldaten* werden die Messwerte (Wasserstand, Temperatur etc.; ➤ Rohdaten) eines Pegels zusammen mit weiteren Informationen (Standort, zuständige Behörde etc.; ➤ Stammdaten) bezeichnet. Im Rahmen dieser Diplomarbeit wird als Messwert lediglich der Wasserstand betrachtet.

Begriff	Erklärung
PEGELONLINE	*PEGELONLINE* ist das gewässerkundliche Informationssystem der Wasser- und Schiffahrtsverwaltung des Bundes. Es stellt tagesaktuelle Messwerte der verschiedenen gewässerkundlichen Parameter (z.B. Wasserstand) der Bundeswasserstraßen zur Verfügung.
PegelSim	*PegelSim* ist die Bezeichnung eines Projektes zur Simulation eines Pegelnetzes. Dieses wurde im Rahmen eines Projektpraktikums an der Universität Koblenz erstellt und dient der Simulation der Pegel für PegelSuite. [BGH+10]
PegelSuite	*PegelSuite* ist die Bezeichnung des Projektes innerhalb der Diplomarbeit. Es beinhaltet das Abrufen und Empfangen von Pegeldaten, die Erstellung von Workflows zur automatisierten Verarbeitung von Pegeldaten eines Gewässers sowie die Orchestrierung der erstellten Web Services durch eine BPEL-Engine.
Plausibilisierung	Die *Plausibilisierung* bezeichnet das Überprüfen der Pegeldaten anhand eines numerischen Abflussmodells. Die Daten werden zudem mit einem Qualitätsmerkmal versehen.
Pull-Betrieb	Im *Pull-Betrieb* werden die Daten beim Pegel aktiv angefragt. Dabei kann ein Zeitraum angegeben werden, für welchen man Pegeldaten erhalten möchte.
Push-Betrieb	Beim *Push-Betrieb* werden die Pegeldaten in regelmäßigen Abständen aktiv vom Pegel, ohne vorherige Aufforderung, propagiert.
Repräsentation (REST)	Eine *Repräsentation* beschreibt im REST-Kontext den derzeitigen Zustand einer Ressource. Repräsentationen werden für den Transfer zwischen Komponenten verwendet. [Fie00]

Begriff	Erklärung
Representational State Transfer	*Representational State Transfer* (REST) beschreibt einen Architekturstil für Hypermediasysteme [Fie00]. REST wird im Allgemeinen für Web Services über HTTP verwendet.
Ressource (REST)	Im REST-Kontext kann eine *Ressource* jede beschreibbare Information sein. [Fie00]
REST	➤ REpresentational State Transfer
Rohdaten	*Rohdaten* bezeichnen die unbearbeiteten Daten, die ein Pegel gemessen hat.
Serviceorientierte Architektur	Bei der *Serviceorientierten Architektur* (SOA) handelt es sich um ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet werden kann [OAS06].
SOA	➤ Serviceorientierte Architektur
SOAP	*SOAP* ist eine XML-basierte Sprache zur Übertragung von Informationen zwischen Systemen in Verbindung mit Web Services. [W3C00]
SODA	*SODA* ist ein Abrufsystem zur Datenfernübertragung der Firma Kisters. Es wird derzeitig von PEGELONLINE verwendet um die Messstellen abzurufen.
Stammdaten	*Stammdaten* bezeichnen die generellen Informationen über einen Pegel, wie Standort, Behörde etc.
UDDI	➤ Universal Description, Discovery and Integration
UML	➤ Unified Modeling Language
Unified Modeling Language	Die *Unified Modeling Language* (UML) ist eine von der OMG standardisierte Sprache zur Modellierung, Dokumentation, Spezifizierung und Visualisierung der verschiedenen Aspekte von Softwaresystemen. Zu diesem Zweck stellt UML mehrere Diagrammarten zur Verfügung. [RHQ+05]

Begriff	Erklärung
Uniform Resource Identifier	Ein *Uniform Resource Identifier* (URI) ist eine Zeichenfolge, welche der Identifikation von abstrakten oder physischen Ressourcen dient. [BLFM05]
Universal Description, Discovery and Integration	Unter *Universal Description, Discovery and Integration* (UDDI) versteht man einen Verzeichnisdienst für Unternehmen und deren Web Services. [Gun02]
URI	➤ Uniform Resource Identifier
WADL	➤ Web Application Description Language
Web Application Description Language	Die *Web Application Description Language* (WADL) ist eine XML-basierte Sprache zur Beschreibung Web-Applikationen. [W3C09b]
Web Services Business Process Execution Language	Die *Web Services Business Process Execution Language* (WS-BPEL) ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen auf der Basis von Web Services. [OAS07]
WS-BPEL	➤ Web Services Business Process Execution Language
Web Service	*Web Service*s sind die derzeitig populärste Realisierung einer serviceorientierten Architektur. Es handelt es sich dabei um eine Softwarekomponente, welche eine Machine-to-Machine (M2M) Interaktion, mittels auf XML basierender Sprachen, über ein Netzwerk ermöglicht. [W3C04a]
Web Services Description Language	Die *Web Services Description Language* (WSDL) ist eine Beschreibungssprache auf XML-Basis, mit welcher sich die Funktionen, Daten, Datentypen und Austauschprotokolle von Web Services beschreiben lassen. [W3C01]

Begriff	Erklärung
Workflow	Ein *Workflow* ist ein Geschäftsablauf (Prozess) innerhalb der betrieblichen Wertschöpfungskette, also der zeitliche und standortbezogene Ablauf der Bearbeitung von Aufgaben und dem dazu notwendigen Informationsfluss in einer stark strukturierten, arbeitsteiligen Organisation. [Acr]
WSDL	➤ Web Services Description Language
XHydro	*XHydro* ist ein auf XML-basiertes organisationsübergreifendes und wirtschaftliches Austauschformat zur optimierten Übertragung von Pegeldaten. [Bun08]
XML	➤ Extensible Markup Language
XML-Encryption	Der Standard *XML-Encryption* spezifiziert ein Verfahren zur Verschlüsselung von XML-Dokumenten. [W3C02]
XML Schema	➤ XML Schema Definition
XML Schema Definition	Die *XML Schema Definition* (XSD) ist eine XML-basierte Sprache zur Beschreibung von Beschränkungen für XML-Sprachen. [W3C]
XML-Signature	*XML-Signature* ist ein Standard zur Signatur von XML-Dokumenten. [W3C08b]
XSD	➤ XML Schema Definition
Zeitreihe	Als *Zeitreihe* bezeichnet man im Allgemeinen eine über die Zeit aufgetragene Reihe von Werten [Tre10].

Literaturverzeichnis

[Acr] ACRYS CONSULT: *Workflow Management - Workflow Management Systeme*. – http://www.acrys.com/en/PDF/workflow_management.pdf (abgerufen: 16.07.2011)

[BGH+10] BECKER, Marcel ; GAUTERIN, Alvaro ; HEROLD, Michael ; PICHKUROV, Dmitriy ; RASKATOW, Johann ; RICHTER, Heiko ; SAILE, David ; STRAUSS, Sascha: *Benutzerhandbuch - Simulation eines Pegelmessnetzes*. 3. Version, Juli 2010

[BLFM05] BERNERS-LEE, T. ; FIELDING, R. ; MASINTER, L.: *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*, Januar 2005. – http://tools.ietf.org/pdf/rfc3986 (abgerufen: 16.07.2011)

[Blu98] BLUMSTENGEL, Astrid: *Entwicklung hypermedialer Lernsysteme*. Webseite, Juli 1998. – http://dsor-fs.upb.de/~blumstengel/main_index_titel.html (abgerufen: 16.07.2011)

[Bra09] BRAUN, Iris: *SOA - Entwicklung verteilter Systeme auf Basis serviceorientierter Architekturen: 4. Komposition von Web Services*. Vorlesungsfolien (Technische Universität Dresden), Wintersemster 2008/2009. – http://www.rn.inf.tu-dresden.de/lectures/EvSaBSA/v14.pdf (abgerufen: 16.07.2011)

[Bun] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *Motivation zur Entwicklung von XHydro*. Webseite, . – http://www.xhydro.de/motivation.html (abgerufen: 16.07.2011)

[Bun05] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *Die Bundesanstalt für Gewässerkunde*. Info-Broschüre, August 2005. – http://www.bafg.de/cln_015/nn_161886/DE/03__Die_BfG/info-broschuere,templateId=raw,property=publicationFile.pdf/info-broschuere.pdf (abgerufen: 16.07.2011)

[Bun07a] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *XHydro Beispiele*. Beispiel-Dateien, Juni 2007. – `http://www.xhydro.de/documentation/examples.zip` (abgerufen: 16.07.2011)

[Bun07b] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *XHydro UML-Modell*. HTML-Dokumentation, Juni 2007. – `http://www.xhydro.de/documentation/UML.zip` (abgerufen: 16.07.2011)

[Bun08] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *XHydro - Standardisierter Datenaustausch*. Poster, Februar 2008. – `http://www.bafg.de/cln_030/M1/DE/03__Arbeitsbereiche/02__Messnetze/Hydrometrische_20Messnetze/xhydro__poster,templateId=raw,property=publicationFile.pdf/xhydro_poster.pdf` (abgerufen: 16.07.2011)

[Bun09] BUNDESANSTALT FÜR GEWÄSSERKUNDE: *XHydro-Schema*. XML-Schema-Dateien, September 2009. – `http://www.xhydro.de/documentation/schema.zip` (abgerufen: 16.07.2011)

[Cer02] CERAMI, Ethan: *Web Services Essentials*. 1. Edition. O'Reilly, 2002

[DW09] DAVIS, Mark ; WHISTLER, Ken: *Unicode Collation Algorithm*. Revision 20, Oktober 2009. – `http://www.unicode.org/reports/tr10` (abgerufen: 16.07.2011)

[Ebe08] EBERLE, Hanna: *Konzeption und Implementierung einer Erweiterung des BPEL Standards zur Modellierung und Ausführung kontextbezogener Workflows*, Universität Stuttgart, Diplomarbeit, März 2008. – `http://elib.uni-stuttgart.de/opus/volltexte/2009/3915/pdf/DIP_2748.pdf` (abgerufen: 16.07.2011)

[Fie00] FIELDING, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000. – `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` (abgerufen: 16.07.2011)

[Gun02] GUNZER, Hartwig: *Introduction to Web Services*. März 2002. – `http://archive.devx.com/javasr/whitepapers/borland/12728JB6webservwp.pdf` (abgerufen: 16.07.2011)

[Kd07a] KISTERS AG ; DISY INFORMATIONSSYSTEME GMBH: *XHydro XML-Pegeldaten_Schema*. Version 1.0.2, Juni 2007. – `http://www.xhydro.de/documentation/XHydro_UML_PegelDaten_Schema_V1.0.2.pdf` (abgerufen: 16.07.2011)

[Kd07b] KISTERS AG ; DISY INFORMATIONSSYSTEME GMBH: *XHydro-XML-Schema Dokumentation*. Version 1.3, Juni 2007. – `http://www.xhydro.de/documentation/XHydro_XML_Schema_Dokumentation_V1.3.pdf` (abgerufen: 16.07.2011)

[Kd07c] KISTERS AG ; DISY INFORMATIONSSYSTEME GMBH: *Zeitreihenmodell zur Pegeldatenübertragung*. Präsentation, Juni 2007. – `http://www.xhydro.de/workshop/2/XHydro_time_series_Model2_workshop.ppt` (abgerufen: 16.07.2011)

[KE06] KEMPER, Alfons ; EICKLER, André: *Datenbanksysteme - Eine Einführung*. 6. Auflage. Oldenbourg Verlag, 2006

[McK07] MCKENDRICK, Joe: *IBM acknowledges bypassing UDDI; calls for new SOA registry standard*. Webseite, April 2007. – `http://blogs.zdnet.com/service-oriented/?p=864` (abgerufen: 16.07.2011)

[OAS06] OASIS: *Reference Model for Service Oriented Architecture*. Version 1.0, Oktober 2006. – `http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf` (abgerufen: 16.07.2011)

[OAS07] OASIS: *Web Services Business Process Execution Language*. Version 2.0, April 2007. – `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf` (abgerufen: 16.07.2011)

[Ora10] ORACLE: *MySQL 5.0 Reference Manual*. Revison 21751, Juli 2010. – `http://dev.mysql.com/doc/refman/5.0/en/index.html` (abgerufen: 16.07.2011)

[Pro03] PROKOPCZYK, Malwina: *Multimedia - Eine Definition*. Januar 2003. – `http://ddi.cs.uni-potsdam.de/Lehre/HypermediaLernsystemeWS2002-03/Papers/multimedia_def.pdf` (abgerufen: 16.07.2011)

[RHQ+05] RUPP, Chris ; HAHN, Jürgen ; QUEINS, Stefan ; JECKLE, Mario ; ZENGLER, Barabara: *UML 2 glasklar*. 2. Auflage. Hanser Verlag, 2005

[Sch07] SCHRICKER, Florian: *Projekt "Pegelstand-Darstellung" mit WaterViz*, Universität Koblenz-Landau, Diplomarbeit, 2007

[Sch08] SCHWARZ, Florian: *Webbasierte Anwendungsintegration nach REST-Prinzipien*, Hochschule Augsburg, Diplomarbeit, April 2008. – `http://www.tngtech.com/assets/pdf/AnwendungsintegrationNachREST-20080418.pdf` (abgerufen: 16.07.2011)

[Ste03] STELZER, Roland: *Fuzzy Web - A Web-Based Approach to Fuzzy System Design*. 2003. – http://wiki.atrox.at/images/d/d2/FYP_REPORT.pdf (abgerufen: 16.07.2011)

[Tre10] TREIBER, Martin: *Statistik 2 - Zeitreihen - Abschnitte 16 und 17: Allgemeines; Trendbestimmung mit Regressionsrechnung; gleitenden Durchschnitte und exponentielle Glättung*. Vorlesungsfolien (Technische Universität Dresden), Wintersemester 2009/2010. – http://www.vwi.tu-dresden.de/~treiber/statistik2/statistik_download/folien10.pdf (abgerufen: 16.07.2011)

[Vas07] VASILIEV, Yuli: *SOA and WS-BPEL*. Packt Publishing, 2007

[W3C] W3C: *Schema*. Webseite, . – http://www.w3.org/standards/xml/schema (abgerufen: 16.07.2011)

[W3C00] W3C: *Simple Object Access Protocol (SOAP)*. Version 1.1, Mai 2000. – http://www.w3.org/TR/2000/NOTE-SOAP-20000508 (abgerufen: 16.07.2011)

[W3C01] W3C: *Web Services Description Language (WSDL)*. Version 1.1, März 2001. – http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (abgerufen: 16.07.2011)

[W3C02] W3C: *XML Encryption Syntax and Processing*, Dezember 2002. – http://www.w3.org/TR/2002/REC-xmlenc-core-20021210 (abgerufen: 16.07.2011)

[W3C04a] W3C: *Web Services Architecture*, Februar 2004. – http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ (abgerufen: 16.07.2011)

[W3C04b] W3C: *Web Services Glossary*. Webseite, Februar 2004. – http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211 (abgerufen: 16.07.2011)

[W3C04c] W3C: *XML Schema Part 0: Primer*. 2. Edition, Oktober 2004. – http://www.w3.org/TR/2004/REC-xmlschema-0-20041028 (abgerufen: 16.07.2011)

[W3C04d] W3C: *XML Schema Part 2: Datatypes*. 2. Edition, Oktober 2004. – http://www.w3.org/TR/2004/REC-xmlschema-2-20041028 (abgerufen: 16.07.2011)

[W3C08a] W3C: *Extensible Markup Language (XML) 1.0*. 5. Edition, November 2008. – http://www.w3.org/TR/2008/REC-xml-20081126 (abgerufen: 16.07.2011)

[W3C08b] W3C: *XML Signature Syntax and Processing*. 2. Edition, Juni 2008. – http://www.w3.org/TR/2008/REC-xmldsig-core-20080610 (abgerufen: 16.07.2011)

[W3C09a] W3C: *Namespaces in XML 1.0*. 3. Edition, Dezember 2009. – http://www.w3.org/TR/2009/REC-xml-names-20091208 (abgerufen: 16.07.2011)

[W3C09b] W3C: *Web Application Description Language*, August 2009. – http://www.w3.org/Submission/2009/SUBM-wadl-20090831 (abgerufen: 16.07.2011)

[Wal07] WALTER, Klaus-Dieter: *Machine-to-Machine (M2M)*. Whitepaper, März 2007. – http://www.m2m-alliance.de/uploads/media/Whitepaper.pdf (abgerufen: 16.07.2011)

[Web06] WEB SERVICES INTEROPERABILITY ORGANIZATION: *Basic Profile*. Version 1.1, April 2006. – http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html (abgerufen: 16.07.2011)