Originally published as

Theisselmann, F.; Haubrock, S.; Vorogushyn, S.; Dransch, D. (2006): NaDiNe-Deich - ein Geoinformationsservice für das Katastrophenmanagement. - In: Clemen, Ch. (Eds.), Entwicklerforum Geoinformationstechnik 2006: Junge Wissenschaftler forschen., Shaker, 150.

NADINE-DEICH
-
EIN GEOINFORMATIONSSERVICE FÜR DAS KATASTROPHENMANAGEMENT

Falko Theisselmann[1], Sören Haubrock[2], Sergiy Vorogushyn[2], Doris Dransch[1,2]

[1]Geographisches Institut HU Berlin

[2]GeoForschungsZentrum Potsdam

Kurzfassung:

Wissenschaftliche Modelle werden im Rahmen des Katastrophenmanagements in der Regel als monolithische Anwendungen zur Simulation von Schadensszenarien eingesetzt. Um die Simulationsergebnisse mit anderen Informationen zu verknüpfen, ist die Integration in eine Dienstearchitektur durch Bereitstellung von öffentlichen Schnittstellen für das Modell erforderlich.

Im Rahmen dieser Arbeit wird ein Ansatz konzipiert und umgesetzt, um ein Deichbruchmodell zur Lokalisierung möglicher Überschwemmungsstellen als Webservice verfügbar zu machen, und dieses mit etablierten Diensten in Form eines Workflows zu verknüpfen. Neben einem mit SOAP ansprechbaren WebService zur Kapselung des Simulationsmodells werden u.a. ein WebFeatureService (WFS) zur Datenhaltung sowie ein WebMapService (WMS) zur Generierung der Ergebniskarten in die mit BPEL modellierte Workflow-Architektur integriert.

1 Daten und Modelle im Hochwassermanagement

Die zerstörerische Kraft von extremen Naturereignissen wie Hochwasser führt in zunehmendem Maße zum Verlust von Menschenleben sowie zu materiellen Schäden. Mit der weltweiten Zunahme von Naturkatastrophen wird ein abgestimmtes Katastrophenmanagement immer wichtiger.

Geodaten und -services spielen hierbei eine wichtige Rolle. Mit Hilfe von Hintergrunddaten, vor allem aber mit in Echtzeit generierten Informationen, können kritische Entscheidungen auf ein sachliches Fundament gestellt werden.

1.1 Natural Disasters Networking Platform NaDiNe

Im Rahmen der *Natural Disasters Networking Platform (NaDiNe)*, einem Gemeinschaftsprojekt der vier Helmholtz-Zentren *Alfred-Wegener Institut für Polar- und Meeresforschung (AWI)*, *Deutsches Zentrum für Luft- und Raumfahrt (DLR)*, *GeoForschungsZentrum Potsdam (GFZ Potsdam)* und *GKSS Forschungszentrum,* wird als zentraler Bestandteil ein webbasiertes Kommunikations- und Informationsportal aufgebaut. Die Plattform bündelt verfügbare Expertise, Datenbestände und Aktivitäten und ermöglicht deren synergetische Nutzung. Krisenrelevante Daten werden dabei gemeinsam analysiert und ausgewertet. Wichtige Information kann somit im Krisenfall den Nutzern schnell zur Verfügung gestellt werden.

1.2 Verfügbarkeit von Datensätzen

Im Rahmen von Naturkatastrophen sind vor allem wissenschaftliche Simulationsmodelle sehr wertvolle Instrumente, um Risiken zu analysieren und Gefahren vorzubeugen. Diese Modelle sind bereits in großer Zahl in den genannten wissenschaftlichen Einrichtungen vorhanden und erlauben die Berechnung von Szenarien auf Grundlage von oft lokal gespeicherten Datensätzen. Gerade im Katastrophenmanagement ist die Nutzung von Daten und Simulationsmodellen in Echtzeit essentiell, wird aus verschiedenen Gründen aber selten in effizienter Weise realisiert. Ein wesentlicher Grund ist die monolithische Implementierung der Modellierungssoftware bzw. des proprietären Zugriffs auf Daten und der damit einhergehenden mangelhaften Bereitstellung von nutzbaren Schnittstellen. Um Simulationsmodelle im Ereignisfall verfügbar zu machen und flexibel in andere Kontexte einzubinden, fehlen zurzeit noch geeignete Umsetzungen.

1.3 Zielstellung

Ziel dieser Arbeit ist es, einen Ansatz aufzuzeigen, um die angesprochenen Defizite in der Bereitstellung von wissenschaftlichen Simulationsmodellen zu überwinden. Anhand eines Deichbruchmodells wird dabei eine Service-Orientierte Architektur (SOA) aufgebaut (genannt *NaDiNe-Deich*). Durch Modellierung des Workflows bestehend aus den

entscheidenden Geschäftsprozessen im Katastrophenfall werden die Möglichkeiten dieser Methodik aufgezeigt und diskutiert.

Das hier entwickelte Modell wird schließlich im Rahmen von NaDiNe prototypisch umgesetzt und getestet.

2 Entwicklung einer Service-Orientierten Architektur (SOA)

2.1 Komponenten

NaDiNe-Deich wurde als SOA konzipiert und implementiert. Es besteht im Wesentlichen aus Servicekomponenten und einem Workflow, der die Services koordiniert.

Ziel des implementierten Workflows ist die Bereitstellung einer Karte mit den Simulationsergebnissen des Deichbruchmodells, welches ein vom Nutzer definiertes Szenario simuliert. Dieses Szenario wird bestimmt durch vom Nutzer festgelegte Eingabeparameter für das Deichbruchmodell.

Der technische Teil des Gesamtsystems setzt sich aus verschiedenen Komponenten zusammen. Diese Komponenten werden auf Basis von Internettechnologien angeboten, somit sind die zentralen Komponenten verschiedene Webservices. Ergänzt werden diese durch ein Userinterface, zwei Datenbanken und eine Programm-Bibliothek (shared library), die die Funktionen des Deichbruchmodells enthält. Die Komponenten des Systems sind in Abbildung 1 dargestellt.

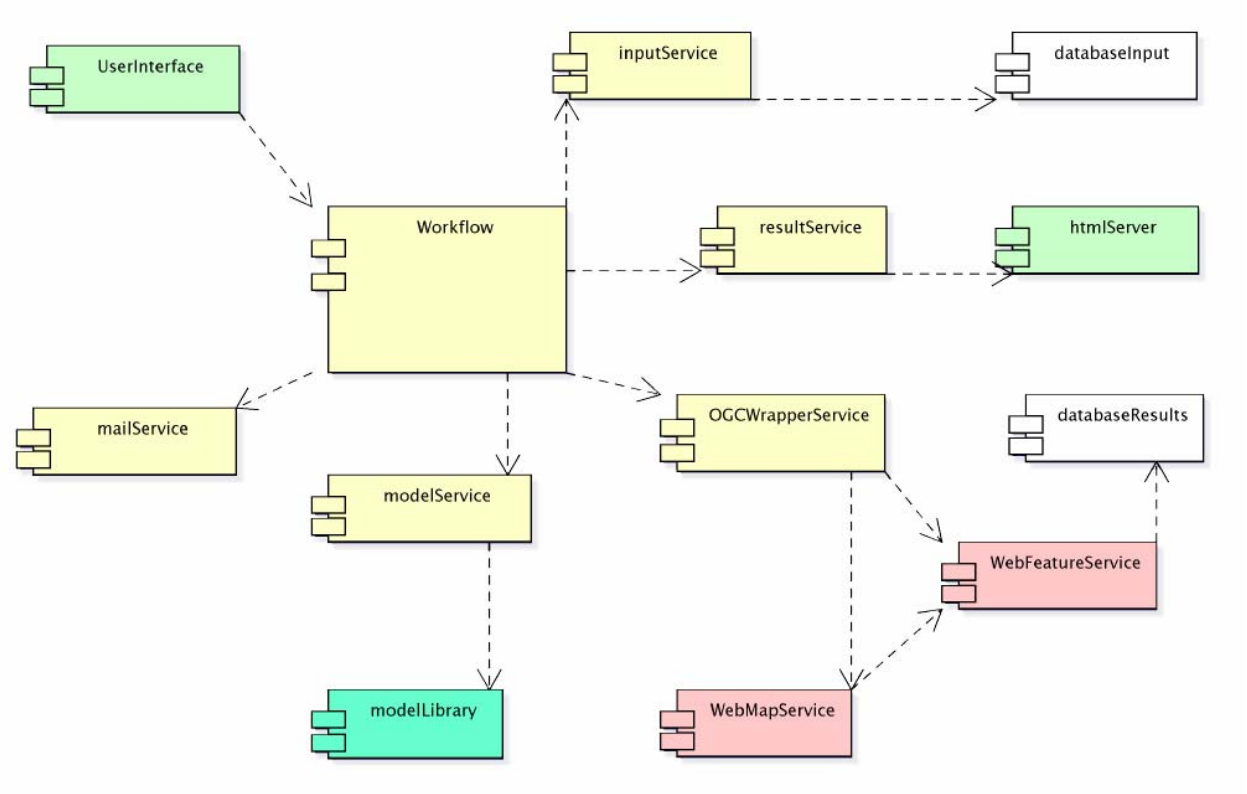

Abb. 1: Komponenten des Informationssystems

Die Komponenten bilden den physischen Aufbau des Informationssystems ab, d.h. sie existieren real und können theoretisch vom System entfernt oder ihm hinzugefügt werden. Zentraler Bestandteil ist hierbei der *Workflow*-Service, der als Steuerungskomponente den Ablauf koordiniert.

Die hier in gelb dargestellten Dienste (*Workflow*, *inputService*, *resultService*, *OGCWrapperService*, *modelService* und *mailService*) sind im Rahmen dieser Arbeit als Standard-WebServices (Zugriff über SOAP) implementiert worden und somit von der Workflow-Engine aus ansprechbar. Die roten Komponenten (*WebMapService* und *WebFeatureService*) entsprechen der Standardisierung nach OGC, sind damit allerdings keine SOAP-WebServices. Alle übrigen Komponenten sind nicht-standardisierte Entwicklungen (zum Teil Hilfskomponenten).

2.2 Workflow: the „big picture“

In dem hier vorgestellten Anwendungsfall lässt der Benutzer das System einen vorher definierten Arbeitsablauf interaktiv durchführen.

Der Ablauf des implementierten Workflows kann folgendermaßen zusammengefasst werden:

1. Der Experte startet eine Simulation,
2. das Ergebnis wird abgespeichert und eine kartografische Darstellung generiert,
3. der Experte erhält eine Karte mit Deichbruchstellen und Zugriff auf Basisdaten.

Aufbauend auf diesem einfachen Arbeitslauf wurden unter Berücksichtigung technischer Aspekte die Komponenten des Informationssystems festgelegt. Im Wesentlichen wurden hierzu den verschiedenen Aktivitäten einzelne Hauptkomponenten zugeordnet, die anschießend durch notwendige Hilfskomponenten ergänzt wurden (Tabelle 1).

Tab. 1: Zuordnung der einzelnen Aktivitäten zu Komponenten

Aktivität	Komponente
Experteneingabe	UserInterface
Modellauf	modelService
Modellergebnis speichern	WebFeatureService
Karte erzeugen	WebMapService
Ausgabe erzeugen	ResultService
Inputmetadaten aktualisieren	InputService
Experten benachrichtigen	mailService/UserInterface

In Abbildung 2 wird der Ablauf des Workflows in Form eines Sequenzdiagramms skizziert.

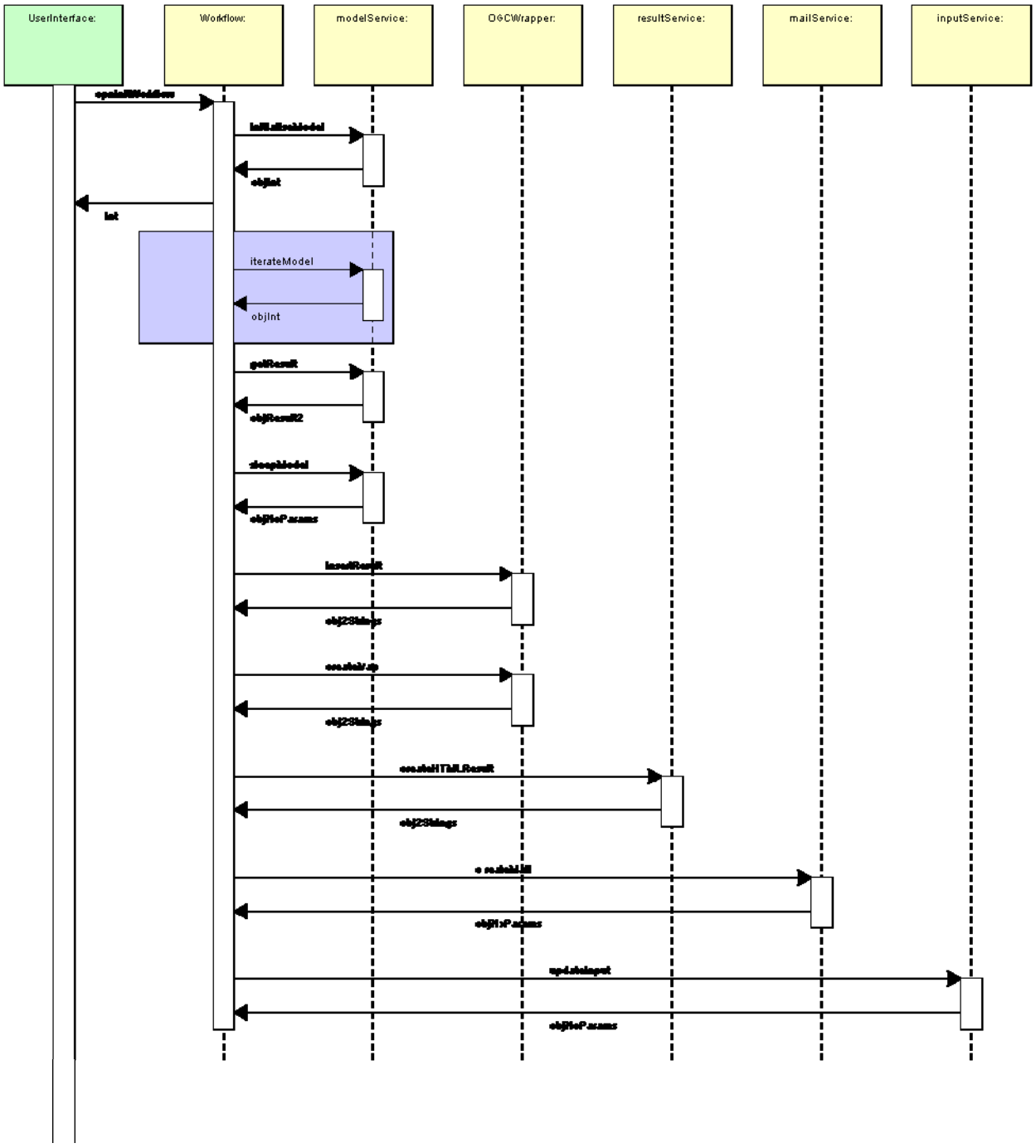

Abb. 2: Sequenz der Nachrichten im Workflow.

Über die Komponente *UserInterface* werden vom Nutzer die Daten für das Szenario ausgewählt (Zeitpunkt, Flussabschnitt). Mit diesen Parametern wird der *Workflow* angesprochen und verwaltet den weiteren Arbeits- und Datenfluss. Der *InputService* bezieht die Informationen zu den abgelegten Inputdaten (Metadaten) zur Initialisierung des Deichbruchmodells aus einer Datenbank. Diese Auswahl wurde durch das *UserInter-*

face vom Nutzer übergeben. Im Anschluss werden die Modell-Eingangsdaten an das Modell übertragen. Der entwickelte *ModelService* kapselt das bereits existierende Deichbruchmodell mit einer WebService-Schicht. Mit Hilfe des *OGCWrapperServices* werden die Ergebnisdaten anschließend in einem *WebFeatureService* (WFS) abgelegt und auf dessen Grundlage über den *WebMapService* (WMS) eine Karte als Ergebnis der Simulation generiert. Der *ResultService* erstellt mit dem erzeugten Bild eine Webseite, bevor der *mailService* eine Benachrichtigung über den abgeschlossenen Workflow an den User mitsamt der URL zum Abrufen des Ergebnisses sendet.

2.3 Implementierung der *Simulationsmodell*-Komponente

Das Simulationsmodell stellt die zentrale Komponente in der Architektur dar, da es entscheidungsrelevante Information im Katastrophenfall Hochwasser generiert. In diesem Fall wird ein hydraulisches Modell mit dem Deichbruchmodell verknüpft und als gemeinsamer Service nach außen angeboten.

2.3.1 Hydraulisches und Deichbruchmodell

Das hydraulische Modell simuliert den Zustand eines Gerinnes (hier: Fluss) in einem bestimmten Gerinneabschnitt für einen festgelegten Zeitraum.

Dieser Zustand wird durch verschiedene Parameter beschrieben, wie z.B. Wasserstände oder Abflüsse. Der Zeitraum eines Modelllaufes wird durch eine Reihe von Modell-Zeitpunkten dargestellt, die mit dem Anfangszeitpunkt beginnen und mit dem Endzeitpunkt enden. Das Intervall zwischen Modell-Zeitpunkten ist die Schrittlänge des Modells.

Während eines Modelllaufes wird mit Hilfe der Modellalgorithmen die Ausprägung der Parameter für alle Modell-Zeitpunkte und alle Modellknoten ermittelt. Die Parameter des Modells beschreiben den Zustand des Modells an einem Modell-Zeitpunkt vollständig und haben an jedem Modell-Zeitpunkt einen bestimmten Wert.

Die Ermittlung von Deichbruchstellen erfolgt im Wesentlichen auf Basis des errechneten Wasserstandes und der Höhe des Deiches am entsprechenden Modellknoten. Hierzu wird jedem Deichpunkt der Wasserstand des entsprechenden Modellknotens zugeordnet. Zu jedem Modell-Zeitpunkt findet eine Überprüfung auf Deichbruchstellen statt.

Um eine Simulation durchführen zu können werden folgende Daten benötigt:

- eine Zeitreihe mit Durchflüssen am obersten Modellknoten,
- der Abfluss am untersten Modellknoten,
- die Geometrie des Flussbettes an den Modellknoten in Form von Querprofilen,
- die Geometrie der Deiche an den Deichpunkten.

Die Schrittlänge kann variabel festgelegt werden. Eine große Schrittlänge hat den Vorteil, dass der Modelllauf schnell durchgeführt werden kann, jedoch birgt dies das Risiko der numerischen Instabilität. Numerische Instabilität hat zur Folge, dass Berechnungen möglicherweise nicht korrekt durchgeführt werden können und der Modelllauf folglich abbricht oder falsche Ergebnisse liefert. Eine geringe Schrittlänge vermindert dieses Risiko, hat aber den Nachteil längerer Rechenzeiten. Beide Aspekte müssen im Rahmen einer SOA berücksichtigt werden.

2.3.2 Implementierung des Modell-Webservice

Im vorliegenden Fall liegt das Modell in FORTRAN vor, während die Services auf Basis von Java angeboten werden. Dies macht eine Integration der verschiedenen Technologien erforderlich.

Diese Integration wurde mit Hilfe eines Wrappers umgesetzt. Der Wrapper verbirgt die Technologie der ihm zugrunde liegenden nativen Softwarekomponente und stellt die Funktionen in einem anderen Format zur Verfügung. Die Aufgabe dieser Wrapper-Komponente ist es, das FORTRAN-Modell durch einen Java-Webservice zur Verfügung zu stellen.

Eine besondere Rolle bei der Implementierung des Wrappers spielt der Informationsfluss zwischen dem Wrapper und der nativen Software. Im vorliegenden Fall wurde diese Kommunikation mit Hilfe des *Java Native Interface (JNI)* und Cross-Compiling implementiert. Das JNI ist eine Programmierschnittstelle um Java-Methoden und die Java Virtual Machine in native Anwendungen einzubetten (SUN, 2006).

JNI ermöglicht unter anderem den gemeinsamen Zugriff auf Variablen durch ein C/C++ Programm und ein Java-Programm. Da C/C++ und FORTRAN Maschinencode erzeugen, ist es möglich, dass diese bei Verwendung geeigneter Compiler und Linker (Cross-Compiling) auf gemeinsame Variablen zugreifen (vgl. Intel, 2006). Bezieht man die Kommunikation der Servicekomponente mit anderen Services ein, so erfordert die Integration des Modells einen Informationsfluss zwischen den Technologien Fortran – C/C++ - Java - XML (Abb. 3).

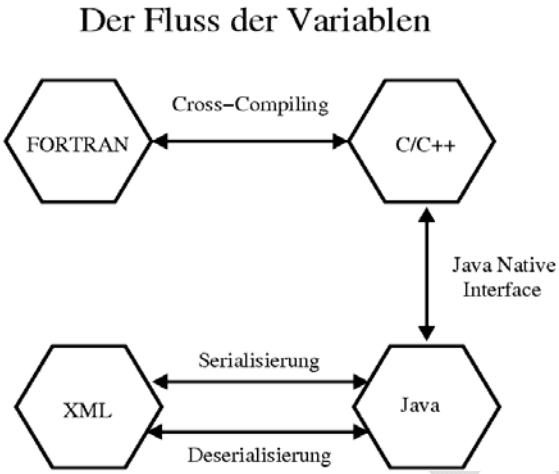

Abb. 3: Variablen-Fluss im Simulations-Modell

Im Folgenden werden die wichtigsten Punkte dieses Weges der Informationen kurz beschrieben.

2.3.2.1 FORTRAN-C/C++

Um FORTRAN und C/C++ zu integrieren wurde in zwei Stufen vorgegangen. Zuerst wurden die zwei Programmteile mit Hilfe des jeweiligen Compilers (gcc bzw. Intel Fortran Compiler for Linux) vom Quellcode in Objektcode übersetzt. Ein Programmteil enthält das Modell und der andere einen C/C++ - Wrapper, der die Funktionen des Modells aufrufen kann. Der entstandene Objektcode konnte anschießend von einem Linker (hier. Intel Fortran Compiler for Linux) in eine gemeinsame ausführbare Komponente überführt werden. Diese Komponente ist im vorliegenden Fall eine Programmbibliothek.

2.3.2.2 C/C++-Java

Der Informationsfluss zwischen C/C++ und Java wurde mit Hilfe des Java Native Interface (JNI) durchgführt. Bei der Verwendung von JNI mussten zwei wichtige Aspekte beachtet werden. Java kann Speicher nur gemeinsam mit C/C++ verwenden, wenn die entsprechenden Variablen vom Java-Typ „Referenztyp“ sind. Zu den Referenztypen gehören Objekte, Strings und Arrays (Krüger, G., 2003). Um Variablen des Modells für den Java-Wrapper sichtbar zu verändern wurde deshalb für alle ausgetauschten Variablentypen eine eigene Java Klasse angelegt, die ein Feld des entsprechenden Java Basistypen enthält. Während der Programmlaufzeit wurden anstelle der „bloßen“ Variablen Objekte ausgetauscht, die die entsprechenden Variablen enthalten.

Diese Objekte konnten mit Hilfe von JNI-Methoden erzeugt und verändert werden, so dass ein Datenaustausch der nativen Modell-Anwendung mit dem Java-Wrapper möglich ist.

2.3.2.3 Java-XML

Die Übertragung der Daten von Java in XML wurde durch *axis* (axis, 2006) und dessen Serialisierungsfunktionalität durchgeführt. Diese Funktionalität übersetzt Java-Objekte in eine entsprechende serialisierte SOAP-XML-Repräsentation. Diese XML-Repräsentation folgt dem SOAP-Standard, ist somit mit WSDL und BPEL kompatibel und eignet sich zur direkten Integration in einen BPEL-Workflow. Durch Deserialisierung kann die XML-Repräsentation wieder in ein Java-Objekt umgewandelt werden.

Der Informationsfluss zwischen den einzelnen Technologien spielt eine entscheidende Rolle für die Funktionalität der Modellkomponente. Fehler bei der Übergabe der Informationen können Modellläufe hinfällig machen.

Die Weiterleitung von Fehlermeldungen erfolgt auf Basis der Rückgabewerte der Funktionen (0 = Erfolg, Andere Werte zeigen einen Fehler an) und mit Hilfe einer kurzen textuellen Beschreibung.

Um die Kommunikation zu erleichtern (vgl. Variablen) wurden alle Zustandsvariablen als Basistypen (Ganzzahl, Gleitkommazahl) oder eindimensionale Felder aus diesen realisiert.

2.3.2.4 SOAP-Anfrage an *modelService*

Eine Anfrage an den *modelService* erfolgt über das SOAP-Protokoll (w3c, 2006). In dem folgenden Listing sind so eine Anfrage und die Antwort des Webservices nach erfolgreicher Beendigung zusammengestellt.

SOAP-Anfrage

für Simulationslauf

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <soapenv:Body>
       <iterateModel xmlns="http://soapinterop.org">
         <stepWidth xsi:type="xsd:int" xmlns="">1000</stepWidth>
       </iterateModel>
     </soapenv:Body>
</soapenv:Envelope>
```

SOAP-Antwort

nach erfolgreicher Simulation

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <iterateModelResponse xmlns="http://soapinterop.org">
        <iterateModelReturn>
          <intValue>1000</intValue>
          <message>The iteration finished.</message>
```

```
            <errCode>0</errCode>
          </iterateModelReturn>
        </iterateModelResponse>
      </soapenv:Body>
</soapenv:Envelope>
```

3 Fazit

Im Rahmen dieser Arbeit wurden verschiedene existierende Komponenten durch Webservices gekapselt und in eine service-orientierte Architektur überführt. Mit Hilfe der implementierten Wrapper konnte gezeigt werden, dass ein existierendes Simulationsmodell für die Benutzung in einem Workflow, der mit standardisierten Technologien ansprechbar ist, nach außen verfügbar gemacht werden kann.

Da das hydraulische und das Deichbruchmodell eng an die Geodaten gekoppelt sind, auf denen die Simulation beruht (z.B. Fließgewässer), stellte sich eine vollständige Entkopplung von Modell und Daten als nicht sinnvoll heraus. Um einen Kommunikations-Overhead zwischen Modell und Datenbank zu vermeiden, beinhalten die genannten Modelle die Daten somit bereits implizit. Dieser Aspekt ist für eine wirkliche lose Kopplung und allgemeine Nutzbarkeit des Modells auch mit anderen Daten bzw. in anderen geographischen Bezügen noch durch geeignete Technologien zu lösen.

Auf Basis der flexiblen service-orientierten Architektur ergibt sich Möglichkeit zur Integration und Nutzung weiterer Daten und Dienste im Katastrophenfall. In einem nächsten Schritt soll die umgesetzte Workflow-Architektur um einen WebService zur Bereitstellung von Echtzeit-Pegeldaten erweitert werden, um somit neben vordefinierten Szenarien auch aktuelle Informationen in die Simulation zu integrieren.

Literatur

[1] Intel, Intel Fortran Compiler Documentation, http://www.intel.com/software/products/compilers/flin/docs/main_for/, 2006, (letzter Zugriff: 9.04.06).

[2] Krüger, G., Handbuch der Java-Programmierung, Addison Wesley Longman, Inc, 2003.

[3] w3c, SOAP-Version 1.2, http://poseidon.home.tlink.de/w3c/REC-soap12-part0-20030624-de_DE/ , 2006, (letzter Zugriff: 14.6.06).

[4] SUN, Java SDK 2, Standard Edition Documentation – Version 1.3.1, http://java.sun.com/j2se/1.3/docs/guide/jni/index.html, 2006, (letzter Zugriff: 9.04.06).

[5] axis, Axis User's Guide, http://ws.apache.org/axis/java/user-guide.html, 2006, (letzter Zugriff: 16.6.06) .