Eberhard Karls
Universität
Tübingen

Integration von Skriptsprachen in BPEL

am Beispiel von PHP und WebSphere Process Server

Diplomarbeit

Volker Klaeren

23. Mai 2007

Lehrstuhl für Technische Informatik
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften
Universität Tübingen

Betreuer (Universität Tübingen)
Prof. Dr. Wilhelm Spruth
Technische Informatik
Wilhelm-Schickard-Institut für Informatik
Sand 13
D-72076 Tübingen

Betreuer (IBM Deutschland Entwicklung GmbH)
Dipl.-Ing. (BA) Martin Smolny
Business Process Solutions Development 4
Schönaicher Straße 220
71032 Böblingen

Zusammenfassung

Mittels der Business Process Execution Language (BPEL) können durch die so genannte Orchestrierung von Service-Aufrufen Lösungen für komplexe Problemstellungen bereitgestellt werden. Um die eigentliche Funktionalität zu erbringen, wird während der Ausführung von BPEL-Geschäftsprozessen auf externe Services zurückgegriffen. Jeder dieser Service-Aufrufe verursacht jedoch einen signifikanten Overhead für das Marshalling der Aufruf-Parameter. Um unnötige Service-Aufrufe während der Ausführung eines BPEL-Geschäftsprozesses zu vermeiden, wurde von der Firma IBM eine Erweiterung für BPEL vorgesehen, die es erlaubt, Java-Code-Fragmente, so genannte 'Java-Snippets', in die Prozessdefinition zu integrieren. Durch diese Java-Snippets sollen einfache Arbeiten zur Laufzeit eines Prozesses ohne Service-Aufruf erledigt werden können.

Durch die Integration von Java-Snippets in BPEL entsteht jedoch eine unerwünschte Abhängigkeit auf die Programmiersprache Java. Diese Abhängigkeit kann durch die Integration alternativer Programmiersprachen gemindert werden. In der vorliegenden Arbeit wird untersucht, welche Programmiersprachen sich für die Integration in BPEL eignen. Am Beispiel der Skriptsprache PHP und IBMs WebSphere Process Server wird anschließend bewertend verglichen, welche Möglichkeiten es gibt, Skriptsprachen zur Laufzeit eines Geschäftsprozesses zur Ausführung zu bringen.
Um die Machbarkeit der Integration von Skriptsprachen in BPEL zu demonstrieren, wird außerdem eine Erweiterung zu BPEL definiert, die es ermöglicht, PHP-Code in BPEL-Geschäftsprozesse zu integrieren. Außerdem wird eine prototpyische Erweiterung des WebSphere Process Server entwickelt und vorgestellt, die in der Lage ist, zur Laufzeit eines BPEL-Geschäftsprozesses PHP-Snippets auszuwerten.

Danksagungen

Die vorliegende Arbeit entstand in Kooperation mit der Abteilung Business Process Solutions Development 4 der IBM Deutschland Entwicklung GmbH im Zeitraum vom Oktober 2006 bis April 2007.

Ich möchte mich an dieser Stelle bei Herrn Prof. Dr.-Ing. Wilhelm G. Spruth für die Ermöglichung dieser Arbeit, sowie die freundliche Unterstützung während der gesamten Bearbeitungszeit bedanken.

Besonders bedanken möchte ich mich bei meinem Betreuer Martin Smolny für die direkte und unkomplizierte Betreuung dieser Arbeit. Für die zahlreichen Anregungen, Tips und Gespräche, bei denen Probleme erörtert und das Vorgehen für die Fortführung der Arbeit durchgesprochen wurde – er brachte viel Zeit und Geduld auf.

Für Hilfe, Aufmunterung und Verständnis danke ich meiner Freundin und meinen Geschwistern und ganz besonders meinen Eltern, ohne die ich nie angekommen wäre, wo ich heute bin.

Inhaltsverzeichnis

1 Einleitung und Aufgabenstellung

1.1 Einleitung

Moderne Softwareprogrammierung ist geprägt von dem Bestreben, Softwarepakete modular aufzubauen. Auf diese Weise soll die Wartung sowie die Wiederverwendung von existierendem Programmcode vereinfacht werden. Unter Zuhilfename von Netzwerktechnologien wird dieses Programmierparadigma ausgeweitet und auf einer höheren Ebene erneut umgesetzt: In so genannten Service-orientierten Architekturen werden sämtliche Softwaremodule als Services aufgefasst und über das Netzwerk unabhängig von Hardwareplattform, Betriebssystem und verwendeter Programmiersprache publiziert und angeboten. Auf diese Weise lässt sich die Wiederverwendungsrate deutlich erhöhen, da sich der Bekanntheitsgrad und die Verfügbarkeit einzelner Softwaremodule schlagartig von kleineren Arbeitsgruppen auf ganze Konzerne oder sogar darüber hinaus erweitert. Die Idee der Service-orientierten Architektur ist indes nichts Neues: Bereits mit altbekannten Technologien wie CORBA oder DCOM wurde dieses Konzept umgesetzt. Durch einen neuen XML-basierten Standard – die so genannten Web Services – hat diese Idee in den letzten Jahren allerdings wieder viel neuen Wind in ihre Segel bekommen. Durch allseits anerkannte Standards zur Beschreibung von Web-Service-Schnittstellen und zur Auffindung von und Kommunikation mit Web Services hat sich eine Technologie entwickelt, in die namhafte Unternehmen seit Jahren viel Geld investieren.

Abgesehen von der erwünschten höheren Wiederverwendungsrate gibt es noch einen weiteren zentralen Faktor, der die nicht unerheblichen Investitionen in diese Technologie antreibt: Durch Abstraktion über die Details einer Implementierung soll es möglich sein, ohne spezielle Programmiersprachenkenntnisse allein durch die Verknüpfung von elementaren Service-Aufrufen Lösungen für komplexere Probleme zusammenzustellen. Ein Standard, der sich in diesem Bereich etabliert hat, ist die Business Process Execution

Language – kurz BPEL. Mit Hilfe von BPEL können über das Netzwerk verteilte Web Services koordiniert und in eine feste Ablaufreihenfolge gebracht werden.

1.2 Aufgabenstellung

Wie in Abschnitt 3.3 über die BPEL-Werkzeuge gezeigt wird, können mit Hilfe von Software-Werkzeugen und ohne Programmiersprachenkenntnisse BPEL-Prozesse entworfen werden, die durch die reine Komposition einzelner Web Services Lösungen für komplexere Problemstellungen bereitstellen. Für bestimmte Teilprobleme in BPEL-Prozessen, die noch vorgestellt werden, sind jedoch Programmiersprachenkenntnisse vorteilhaft.
Für diese Teilprobleme wird von der WebSphere Produktfamilie eine proprietäre Erweiterung des BPEL-Standards angeboten, die so genannten 'Java-Snippets'. Während die Java-Snippets die Flexibilität der Prozessdefinition erhöhen, bringen sie jedoch auch Nachteile für den Geschäftsprozess mit sich. Diese Nachteile werden analysiert und mögliche Lösungen präsentiert.
Eine dieser Lösungen besteht in der Integration zusätzlicher Programmiersprachen in BPEL.

Aufgabe dieser Arbeit ist es, zu analysieren, welche Programmiersprachen sich für die Integration in BPEL eignen. Darüber hinaus sollen die unterschiedlichen Möglichkeiten, Skriptsprachen zur Laufzeit eines BPEL-Geschäftsprozesses auszuführen, untersucht und miteinander verglichen werden. Dieser Vergleich wird anhand der Skriptsprache PHP und IBMs WebSphere Process Server durchgeführt. Zum Abschluss der Arbeit soll eine prototypischen Implementierung – ebenfalls bezogen auf das Beispiel PHP und WebSphere Process Server – die Machbarkeit der Integration von Skriptsprachen in BPEL demonstrieren.

1.3 Aufbau der Arbeit

In Kapitel 2 wird der thematische Hintergrund für die Business Process Execution Language vorgestellt. Es wird dargestellt, worum es sich bei der Idee der Service-orientierten Architektur handelt und worin die Vorteile einer Service-orientierten Architektur mit Web Services liegen.

Kapitel 3 klärt, worum es sich bei einem so genannten 'Geschäftsprozess' handelt und inwiefern Geschäftsprozesse mittels BPEL modelliert werden können. Anschließend werden einige zentrale Eigenschaften und Sprachelemente von BPEL sowie Begrifflichkeiten, die mit BPEL zusammenhängen, präsentiert. Außerdem wird eine Übersicht über die Werkzeuge gegeben, die bei der Erstellung, Wartung sowie Ausführung von BPEL-Prozessen verwendet werden können. Zum Abschluss dieses Kapitels wird die so genannte 'Java Snippet Activity' vorgestellt – eine BPEL-Erweiterung, die besondere Relevanz für die Motivation dieser Arbeit hat.

Aufbauend auf den Ausführungen der vorausgegangenen Kapitel wird in Kapitel 4

die Motivation für diese Arbeit konkretisiert: Es wird dargestellt, wozu Skriptsprachen in BPEL verwendet werden können und aus welchem Grund für die weiteren Ausführungen die Skriptsprache PHP gewählt wurde.

Um die Auswertung von PHP-Code zur Laufzeit eines Geschäftsprozesses zu ermöglichen, muss der PHP-Interpreter auf geeignete Weise in die Laufzeitumgebung des Prozesses integriert werden. Für diese Integration sind mehrere unterschiedliche Technologien denkbar, welche in Kapitel 5 analysiert und vergleichend bewertet werden.

In Kapitel 6 schließlich wird ein konkretes Konzept zur Integration von PHP in BPEL entwickelt und umgesetzt. Ausgehend von einer Erweiterung des BPEL-Standards um ein neues Element, werden die nötigen Schritte zur Erweiterung des WebSphere Process Server dargelegt, welche dafür sorgen, dass BPEL-Dokumente, die PHP-Code enthalten als gültig erkannt und korrekt installiert werden. Des Weiteren werden die wesentlichen Schritte aufgezeigt, die dazu nötig sind, PHP-Code zur Laufzeit eines Geschäftsprozesses auszuführen.

Zum Abschluss dieser Arbeit findet in Kapitel 7 eine kurze Zusammenfassung der Ergebnisse dieser Arbeit statt. Des Weiteren wird hier gezeigt, wie eine Migration von PHP-Webanwendungen auf ein BPEL-basiertes Geschäftsprozess-Modell durchgeführt werden kann.

2 Hintergrund

Die Ausführungen in diesem Kapitel führen in das thematische Umfeld dieser Arbeit ein. Sie sollen dabei helfen, BPEL sowie die im Rahmen dieser Arbeit vorgestellte Erweiterung zum BPEL-Standard in den richtigen Kontext zu setzen.
Die Ausführungen in diesem Kapitel stützen sich, wo nicht anders ausgezeichnet, auf die Bücher **Web Services Platform Architecture** [WCL+05] sowie **Service-orientierte Architekturen mit Web Services** [DJMZ05].

2.1 Situationsbeschreibung

Komplexere Softwaresysteme werden heutzutage so gut wie nie von Grund auf neu entworfen. Der Regelfall – zumindest in größeren Unternehmen – ist vielmehr das exakte Gegenteil dessen: Die IT-Infrastruktur größerer Unternehmen besteht aus einer Vielzahl von Softwarekomponenten, entwickelt mit unterschiedlichen Programmiersprachen und verteilt auf eine heterogene Systemlandschaft. Diese Systeme sind geprägt von individuell gestalteten Punkt-zu-Punkt-Verbindungen, die die einzelnen Komponenten sehr eng aneinander koppeln. Sie sind historisch gewachsen und haben das Unternehmen über die Jahre für Entwicklung und Wartung viel Geld gekostet. Die einzelnen Bestandteile haben sich teilweise schon jahrzehnte lang bewährt und der Erfolg des Unternehmens hängt maßgeblich von der korrekten Funktionsweise dieser Systeme ab. Änderungen an diesen Systemen sind daher bei den Unternehmen höchst unbeliebt. Allerdings verschlingen diese riesigen Systeme einen großen Teil des Budgets für die IT-Infrastruktur alleine für deren Wartung. Außerdem sind die Unternehmen permanent auf die Anpassung der Systeme angewiesen, da sie nur mit Systemen, die perfekt auf die aktuelle Marktsituation zugeschnitten sind, konkurrenzfähig bleiben können. Diese Situation wird verschärft durch den aktuellen Trend zu immer kürzeren Produktzyklen und der damit einhergehenden höheren Frequenz von Änderungen an

den Systemen.
Daher wird permanent nach Lösungen gesucht, die vor allem die folgenden Eigenschaften vereinen:

- **Integration**: Einfache Integration anderer Systeme, auch von anderen Herstellern.
- **Flexibilität**: Geänderte Rahmenbedingungen sollten möglichst flexibel abgebildet werden können.
- **Austauschbarkeit**: Komponenten sollten problemlos austauschbar sein.

Um diese Forderungen erfüllen zu können, werden insbesondere standardisierte, öffentliche Schnittstellen sowie eine loose Kopplung zwischen den Komponenten benötigt. Ein Lösung, die diesen Ansprüchen gerecht wird, ist eine auf Web Services basierende Service-orientierte Architektur.
Die beiden Begrifflichkeiten 'Web Service' sowie 'Service-orientierte Architektur' werden im Folgenden einführend vorgestellt:

2.2 Web Services

Web Services stellen eine mögliche Realisierung der Service-Idee dar. Die Idee der Services ist zunächst nur ein abstraktes Konzept, dass auf viele Arten umgesetzt werden kann. Diese Idee wird im Folgenden vorgestellt:

Service-Definition

> „A Service is available at a particular endpoint in the network, and it receives and sends messages and exhibits behavior according to its specification“. [WCL+05]

Diese knappe Forumlierung trifft die wesentlichen Grundzüge eines Services. Ein Service

- ist über das Netzwerk verfügbar,
- sendet und empfängt Nachrichten,
- bietet nach außen ein klar definiertes Verhalten an, welches durch seine Schnittstellenbeschreibung definiert wird

Services bündeln zudem Funktionalität, die sich mit bestimmten thematisch klar abgegrenzten Problemen beschäftigen. Beispielsweise könnte ein Kreditkartenservice die folgende Funktionaltät anbieten: Kreditkartendaten prüfen, Zahlungen mittels Kreditkarte vornehmen oder stornieren und Kreditkarte einziehen. Der Funktionsumfang sowie die erwarteten Ein- und Ausgaben des Services werden in einer Schnittstellenbeschreibung festgehalten. Des Weiteren wird in der Schnittstellendefinition festgehalten, an welcher Stelle im Netzwerk dieser Service verfügbar ist. Die Schnittstellendefinition ist so gehalten, dass sie unabhängig von einer bestimmten Programmiersprache oder Systemplattform ist. Auf diese Weise wird eine Unabhängigkeit zwischen Schnittstellendefinition und Implementation der Services erreicht.

Eigenschaften von Web Services

Die zuvor beschriebenen Services sind, wie bereits erwähnt, ein abstraktes Konzept. Bei Web Services dagegen handelt es sich um eine konkrete Technologie, welche im Folgenden vorgestellt wird. Das World Wide Web Consortium (W3C), welches eine wichtige Rolle bei der Standardisierung von Web Services spielt, definiert Web Services folgendermaßen:

> „A Web service is a software system identified by a URI (RFC 2396), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols“ [ABCG04].

Diese Definition enthält alle wesentlichen Bestandteile, die die Web Service Technolgie ausmacht. Web Services sind danach charakterisiert durch:

- Die XML-basierte Schnittstellenbeschreibung
- sowie Mechanismen zur Auffindung von Web Services und
- die XML-basierte Kommunikation untereinander.

Die einzelnen Bestandteile sind vollständig standardisiert. Die Kommunikation zwischen den Web Services geschieht mit Hilfe des Simple Object Access Protocol (SOAP) und die Schnittstellenbeschreibung wird mittels der Web Services Description Language (WSDL) vorgenommen. Die Auffindung wird mit Hilfe des Universal Description, Discovery and Integration (UDDI) Standards vorgenommen. Das Zusammenspiel der einzelnen Standards ist in Abbildung 2.1 angedeutet und wird im Folgenden oberflächlich vorgestellt. Eine ausführlichere Beschreibung findet sich beispielsweise in [Bur04], [WCL+05]:

- **Publizieren**: Der Service-Anbieter erstellt einen Service und die dazu passende **WSDL**-Schnittstellenbeschreibung. Bei der so genannten 'Service Registriy' – zu deutsch Verzeichnisdienst – wird diese Beschreibung veröffentlicht und die Adresse, unter der der Service erreichbar ist, hinterlegt. Der Verzeichnisdienst ist mittels **UDDI** realisiert.
- **Finden**: Der Service-Nutzer sucht nach einem Service, der die benötigte Funktionalität bietet. Dazu durchsucht er den Verzeichnisdienst anhand passender Kriterien. Der Verzeichnisdienst liefert die entsprechenden Services und stellt die Informationen, die zur Verwendung des Services benötigt werden, zur Verfügung.
- **Binden und Interagieren**: Der Service-Aufrufer erzeugt mit Hilfe der Schnittstellenbeschreibung eine **SOAP**-Nachricht, die den gewünschten Service-Aufruf zusammen mit allen benötigten Parametern enthält. Er sendet diese Nachricht mittels eines vom Service-Anbieter unterstützten Übertragungsprotokolls – typischerweise HTTP über TCP/IP, aber es sind auch andere Protokolle denkbar – an den Service-Anbieter. Der Anbieter empfängt die Nachricht und erkennt, dass es sich um einen Service-Aufruf handelt. Der entsprechende Service-Aufruf wird durchgeführt und das Ergebnis wird ebenfalls in eine SOAP-Nachricht verpackt und an den Aufrufer zurückgesendet.

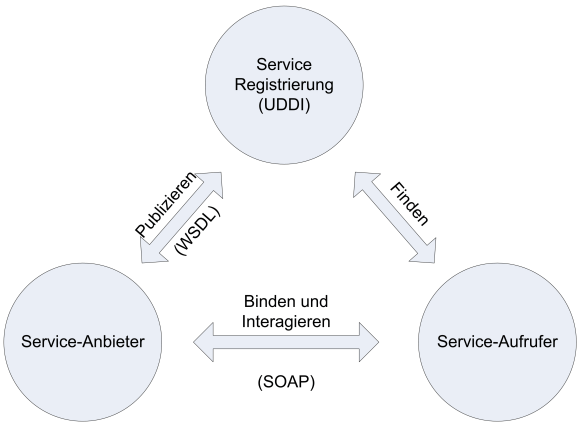

Abbildung 2.1: Komponenten der Web Service Architektur

2.3 Service-orientierte Architekturen

Das Konzept der Service-orientierten Architektur (SOA) ist ein Softwarearchitektur-Paradigma. Es basiert auf der grundlegenden und zugleich altbekannten Idee, komplexe Softwareprodukte durch modularen Aufbau handhabbar zu machen. Diese Komponenten werden in Service-orientierten Architekturen in Form von Services über das Netzwerk angeboten.
Dieser Abschnitt gibt einen kurzen Überblick über die Ziele und Vorteile, die mit einer Service-orientierten Architektur realisiert werden sollen. Ausführlicher mit dem Thema SOA befasst sich beispielsweise [KBS04].

Ziele einer SOA

Zur Kosteneinsparung in der IT-Infrastruktur wird seit vielen Jahren schon auf das Konzept der Service-orientierten Architektur (SOA) gesetzt. Die Eigenschaft „Service-orientiert“ bedeutet dabei nicht wesentlich mehr, als dass die einzelnen Bausteine einer solchen Software-Architektur als Services bezeichnet und über das Netzwerk verfügbar gemacht werden. Durch den Service-orientierten Aufbau einer Software sollen unter anderem folgende Ziele erlangt werden:

- Die Vermeidung von Redundanzen,
- die Erhöhung der Wiederverwendungsrate von Programmcode
- und die Interoperabilität zwischen unterschiedlichen Plattformen.

Unnötige **Redundanzen** bergen grundsätzlich das Risiko von Inkonsistenzen und erschweren die Wartung der Software erheblich. Bei Änderungen der Logik in einer bestimmten Komponente muss die redundante Logik in einer anderen Komponente stets nachgeführt werden. Es ist daher nicht verwunderlich, dass beim Entwurf von

Softwarepaketen versucht wird, Redundanzen, wo immer es möglich ist, zu vermeiden. Mit Service-orientierter Denkweise können Redundanzen auf natürliche Weise vermieden werden. Für jedes Stück Programmlogik gibt es zentral verfügbare Services, auf die im Bedarfsfall zurückgegriffen werden kann.

Nach Bereitstellung einer Infrastruktur von Web Services mit den häufig benötigten Funktionen muss nicht mehr viel getan werden, um die Erhöhung der **Wiederverwendungsrate von Programmcode** zu erlangen. Durch die Bereitstellung einer zentralen Registrierungskomponente für Services (siehe 2.2) erhöht sich die Sichtbarkeit und Verfügbarkeit einzelner Services erheblich. Wird bei der Entwicklung von neuen Softwarekomponenten auf die derart verfügbar gemachten Services zurückgegriffen, werden gleich zwei zentrale Anforderungen an moderne Softwareentwicklung erfüllt: Die Vermeidung von Redundanz und die Erreichung einer hohen Wiederverwendungsrate des Programmcodes. Dem Thema Wiederverwendbarkeit in Service-orientierten Architekturen widmet sich [Hal06].
Ein weiteres wichtiges Ziel, dass durch den Service-orientierten Aufbau einer Software erlangt werden soll, ist es, die **Interoperabilität** zwischen unterschiedlichen Plattformen und Systemen herzustellen. Diese Aussage bezieht sich darauf, dass heutzutage die wenigsten Systeme lediglich auf einer einzigen Technologie basieren. Um am Markt erfolgreich zu sein, ist es eine Schlüsselvoraussetzung, beliebige Systeme – beispielsweise von Zulieferern – ohne großen Aufwand in die eigene IT-Infrastruktur zu integrieren. Dieser Anspruch kann nur von einem System, dass hochgradig interoperabel ist, erfüllt werden.

Beschränkungen der vorhandenen SOA-Ansätze

Diese Ansprüche werden von existierenden Lösungen wie der Common Object Request Broker Architecture (CORBA, siehe [Bol01]) und Microsoft's Distributed Component Object Model (DCOM, siehe [EE98]) erfüllt, jedoch haben diese Ansätze auch einige Beschränkungen [Blo05]:

- Die Lösungen sind **proprietär**: DCOM wurde von Anfang an vollständig von Microsoft kontrolliert. CORBA ist zwar dem Prinzip nach ein standardisiertes Produkt, in der Praxis funktionieren die einzelnen Komponenten jedoch nur, wenn man sich auf die Produkte eines Herstellers beschränkt. Auf diese Weise entsteht eine Bindung an einen Hersteller, was sich negativ auf die Flexibilität des Unternehmens auswirkt. Insbesondere, wenn es darum geht, im Rahmen einer Kooperation, die Services eines anderen Unternehmens zu integrieren, welche mit dem Produkt eines anderen Herstellers realisiert sind.
- Das zweite Problem ist die so genannte **„Enge Kopplung"**: Enge Kopplung bedeutet, dass die Implementierung des Service-Aufrufers mit der Implementierung des Service-Anbieters in engem Kontakt steht. Bei Objekt-orientierten Technologien bedeutet das beispielsweise, dass ganze Objekte eines Service-Anbieters geladen werden, bevor sie zur Ausführung kommen. Um ein bestimmtes Objekt korrekt verwenden zu können, ist ein gewisses Vorwissen über den Aufbau des Objektes unabdingbar. Dieser Umstand führt dazu, dass im Fall einer Änderung

am Objekt des Service-Anbieters für gewöhnlich auch Änderungen an dem Objekt des Aufrufers vorgenommen werden müssen. Weitere Details zum Thema enge Kopplung können aus [KBS04] entnommen werden.

Umsetzung einer SOA mit Hilfe von Web Services

Eine Realisierung einer SOA unter Verwendung von Web Services umgeht die soeben angesprochenen Probleme und zwar aus den folgenden Gründen:

- Web Services sind ein offener Standard. Jeder Hersteller kann in seinen Produkten die Unterstützung für Web Services vorsehen. Die Interoperabilität wird auf diese Weise sichergestellt.
- Web Services sind lose gekoppelt; dadurch wird die Abhängigkeit unter den einzelnen Komponenten auf ein Minimum reduziert.

Wie im nächsten Kapitel gezeigt wird, lassen sich aufbauend auf den im Rahmen einer SOA bereitgestellten Web Services durch einfache Verkettung von elementaren Service-Aufrufen Lösungen für komplexere Probleme bereitstellen.

3

Business Process Execution Language

Die einzelnen Services, die im Rahmen einer service-orientierten Architektur angeboten werden, können als Komponentenmodell auf einem hohen Abstraktionsniveau aufgefasst werden [DJMZ05]. Die Verwendung eines solchen Komponentensystems birgt Vorteile wie höhere Wiederverwendungsrate und das Geheimnisprinzip („Information Hiding Principle“) [Bal96]. Diese Komponenten können nun bei der Lösung von komplexeren Problemen verwendet werden, beispielsweise bei der automatisierten Abwicklung von Geschäftsprozessen, wie sie im folgenden Abschnitt vorgestellt wird.
Eine nahe liegende Idee ist es, zur automatisierten Abwicklung eines Geschäftsprozesses ein Programm zu schreiben, welches im Wesentlichen die einzelnen Komponenten in der richtigen Reihenfolge mit den richtigen Parametern aufruft. Das Problem an dieser Herangehensweise liegt auf der Hand: Soll etwas an der Logik oder Ablaufreihenfolge des Geschäftsprozesses verändert werden, ist grundsätzlich die Einsicht in den Quellcode des Programmes nötig.
Eine bessere Lösung für diese Ablaufkoordinierung stellt die Business Process Execution Language – kurz BPEL – dar, die in diesem Abschnitt vorgestellt wird. Sämtliche Ausführungen in diesem Kapitel beziehen sich auf den BPEL Standard in der Version 1.1 [ACD+03]. Die im Rahmen von BPEL 2.0 [AAA+06] eingeführten Veränderungen am BPEL-Standard haben für diese Arbeit keinerlei Relevanz.
Bevor genauer auf das Thema BPEL eingegangen wird, wird zunächst der Begriff Geschäftsprozess etwas detaillierter beleuchtet:

3.1 Geschäftsprozesse

Sowohl die Produktion von Gütern als auch die Erbringung von Dienstleistungen bestehen für gewöhnlich aus einer Reihe einzelner Aktivitäten. Um ein bestimmtes betriebliches Ergebnis zu erreichen, ist die korrekte Abfolge der einzelnen Aktivitäten

von zentraler Bedeutung. Diese Abfolge wird formal definiert in Geschäftsprozessen. Geschäftsprozesse laufen in einfachen Fällen linear ab, oft jedoch enthalten sie Bedingungen und Schleifen, die den Kontrollfluss festlegen: Abhängig von dem Ausgang der vorausgegangenen Aktivität werden weitere Aktivitäten eventuell übersprungen oder auch wiederholt. Geschäftsprozesse müssen sich dabei nicht zwingend auf ein Unternehmen beschränken, im Gegenteil: Viele Geschäftsprozesse greifen zur Erreichung des vorgegeben Ziels auf Geschäftsprozesse eines anderen Unternehmens zurück.

Beispiel für einen Geschäftsprozess

Ein Beispiel für einen Geschäftsprozess ist das Buchen einer Reise, schematisch dargestellt in Abbildung 3.1[1]:

Zu Beginn des Buchungsprozesses (1) liegen sämtliche für die Buchung einer Reise notwendigen Informationen vor. Diese Informationen umfassen beispielsweise den Namen und die Adresse des Kunden, Kreditkarteninformationen und Informationen über das Reiseziel und die Reisedauer sowie weitere Informationen. Die erste Aktion die ausgeführt werden muss, ist das Überprüfen der Kreditkarteninformationen (2). Sollte die Kreditkartennummer ungültig sein, so werden statt des normalen Arbeitsablaufes entsprechende Aktionen zum Behandeln des Fehlers durchgeführt (3). Beispielsweise die Erzeugung einer entsprechenden Fehlermeldung, welche anschließend dem Kunden präsentiert werden kann. Ist die Kreditkarte gültig, so werden gleichzeitig drei verschiedene Reservierungen vorgenommen (4): Für den Flug, das Hotel sowie einen Leihwagen. Der Vorgang zum Reservieren eines Leihwagens wird dabei solange wiederholt, bis die Reservierung erfolgreich durchgelaufen ist (5). Sobald alle parallelen Aktionen beendet sind, wird überprüft, ob alle Aktionen erfolgreich ausgeführt wurden. Ist dies der Fall, so wird eine Bestätigungsnummer für den Kunden erzeugt, mit der er zu einem späteren Zeitpunkt auf seine Reservierung zurückgreifen kann (6). Anschließend wird das Ergebnis der Reservierung dem Kunden präsentiert (7).

Problem

Der in Abbildung 3.1 dargestellte Ablauf der einzelnen Aktionen könnte so, wie er ist, problemlos von einem Mitarbeiter des Reisebüros ausgeführt werden. Es wäre damit sichergestellt, dass der Mitarbeiter keinen wichtigen Arbeitsschritt vergisst und der Kunde bei jedem Besuch im Reisebüro einen gleichartigen Service erhält. Bei häufigen Buchungsanfragen wäre der Mitarbeiter des Reisebüro jedoch einen Großteil seiner Arbeit mit Routineaufgaben beschäftigt und könnte so seiner wichtigen Beraterfunktion nicht mehr ausreichend Rechnung tragen. Schöner wäre eine vollständig automatisierte Ausführung dieses Prozesses durch ein EDV-System. Diese Automatisierung kann unter der Voraussetzung, dass die einzelnen Arbeitsabschnitte – im Diagramm dargestellt

[1] Dieses Beispiel stammt von der Webseite **Business Process Choreographer samples** http://publib.boulder.ibm.com/bpcsamp/ (zuletzt besucht 26.04.2007)

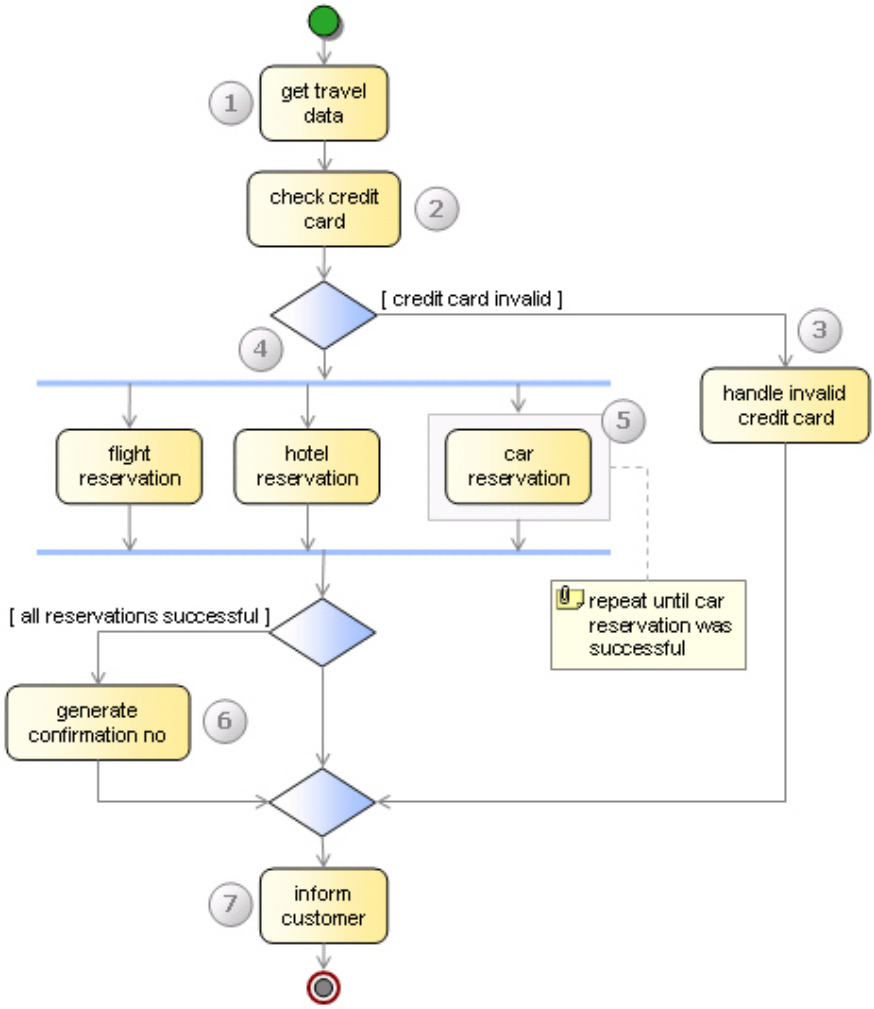

Abbildung 3.1: Flussdiagramm eines Reisebuchungsvorgangs

als rechteckige gelbe Kästchen – als Web Services zur Verfügung stehen, mit Hilfe von BPEL geschehen. In diesem Fall teilt der Mitarbeiter dem System sämtliche relevanten Daten für den Buchungsprozess mit und das System kümmert sich um die Details, wie beispielsweise die korrekte Abfolge der Service-Aufrufe und den Transport der Daten.

3.2 Eigenschaften von BPEL

In erster Linie geht es bei BPEL um die so genannte 'Orchestrierung' von Web Services, dass heißt darum, festzulegen, welche Services in welcher Reihenfolge aufgerufen werden sollen. Des Weiteren kann mittels BPEL der Datenfluss zwischen den einzelnen Service-Aufrufen modelliert werden sowie Maßnahmen definiert werden, die im Fehlerfall durchgeführt werden sollen.

Im Folgenden wird eine knappe Übersicht über einige Sprachelemente von BPEL gegeben, mit denen diese Ziele erreicht werden sollen. Für eine detaillierte Übersicht sei auf [BBC06] verwiesen. Der komplette BPEL-Standard in der Version 1.1 findet sich in [ACD+03]. Ausführlichere Code-Beispiele können beispielsweise einer von IBM veröffentlichen Sammlung von beispielhaften BPEL-Prozessen auf der Webseite **Business Process Choreographer samples** [IBM07] entnommen werden.

3.2.1 Prozess-Struktur

Mit Hilfe von BPEL-Dokumenten sollen Geschäftsprozesse auf standardisierte Weise beschrieben werden. Die Details der BPEL-Prozessbeschreibung sind für diese Arbeit nebensächlich. Der generelle Aufbau eines BPEL-Prozesses ist in Listing 3.1 abgedruckt und wird im Folgenden beschrieben:

Listing 3.1: Aufbau eines BPEL-Prozesses

```
1  <process name="Travelbooking">
2    <!-- hier werden globale Eigenschaften festgelegt -->
3    <variables>..</variables>
4    <faultHandlers>..</faultHandlers>
5    <compensationHandlers>..</compensationHandlers>
6    ..
7
8    <!-- hier folgen die auszuführenden Aktivitäten -->
9    <invoke>..</invoke>
10   ..
11
12 </process>
```

Jedes BPEL-Dokument muss exakt ein `<process>`-Element enthalten, welches zugleich das Wurzelelement des BPEL-Dokumentes darstellt. Des Weiteren muss in jedem Prozess mindestens eine Aktivität definiert sein. Alle weiteren Elemente sind optional.

Zu Beginn der Prozessdefinition können einige globale Eigenschaften des Prozesses festgelegt werden. Einige dieser Eigenschaften sind in den Zeilen Zeilen 1 bis 5 des

Listings 3.1 abgedruckt, eine vollständige Übersicht über sämtliche möglichen Eigenschaften können dem BPEL-Standard [ACD+03] entnommen werden.
Auf die Festlegung der globalen Einstellungen folgt die Definition der einzelnen Aktivitäten, die im Rahmen des Prozesses ausgeführt werden sollen (Zeilen 8ff.).

Die Elemente, die die globalen Eigenschaften des Prozesses beschreiben, werden im Folgenden beschrieben. Eine Auflistung der einzelnen Elemente, die in einem BPEL-Prozess zur Erstellung eines Prozessmodells verwendet werden können, findet sich im nächsten Abschnitt.

`<variables>`
Prozesse sind per Definition zustandsbehaftet. Dieser Zustand wird gehalten durch die ausgetauschten Nachrichten und die im Prozess gespeicherten Daten [DJMZ05]. Die Daten des Prozesses werden in Prozessvariablen gespeichert, welche im Kopf der Geschäftsprozess-Definition definiert werden. Der Beginn und das Ende des Variablen-Deklarationsblockes werden über die Schlüsselwörter `<variables>` respektive `</variables>` signalisiert. Innerhalb dieses Blockes werden einzelne Variablen mit dem Schlüsselwort `<variable>` festgelegt. Alle Variablen sind statisch getypt und können simple Datentypen – wie beispielsweise `xsd:int` oder `xsd:string` – oder aber auch zusammengesetzte komplexe Datentypen, wie beispielsweise `'Kunde'` sein. Ein Beispiel für einen solchen Deklarationsblock ist Listing 3.2 zu entnehmen:

Listing 3.2: Variablendeklaration in BPEL-Prozessen

```
1 <variables>
2   <variable name="einString" type="xsd:string"/>
3   <variable name="Kunde1" type="ns0:Kunde"/>
4 <variables>
```

In diesem Beispiel werden zwei Variablen festgelegt: Eine Variable vom Typ `xsd:string` mit Namen `einString` sowie eine Variable mit dem komplexen Typ `Kunde` und dem Namen `Kunde1`. Mehr Details über die verschiedenen Datentypen werden in Abschnitt 3.2.3 vorgestellt.

`<faultHandlers>,<compensationHandlers>`
Eine wichtige Eigenschaft von BPEL ist die Unterstützung für langlaufende transaktionale Prozesse. Mit Hilfe des Konstruktes `<faultHandlers>` kann definiert werden, welche Aktionen im Falle eines Fehlers durchgeführt werden sollen. Beispielsweise könnte eine Aktion erneut ausgeführt werden. Es ist aber auch möglich festzuschreiben, wie im Falle eines dauerhaften Fehlers reagiert werden soll: Mit Hilfe des Konstruktes `<compensationHandlers>` wird definiert, welche Aktionen zur Kompensation bereits vollständig durchgeführter Transaktionen ablaufen müssen. Der Hintergrund dafür ist der Folgende:
Vollständig durchgelaufene Transaktionen können nicht mehr durch ein einfaches 'rollback' rückgängig gemacht werden. Stattdessen sind neue Transaktionen nötig, die versuchen, die vorausgegangenen Transaktionen rückgängig zu machen. Details zur Kompensation können dem Artikel **'Extending the concept of transaction compensation'** [CGV+02] entnommen werden.

3.2.2 Prozess-Aktivitäten

Der zentrale Teil eines BPEL-Prozesses dreht sich um die Koordination und richtige Ausführung von Einzelaktivitäten. Der BPEL-Standard definiert zwei unterschiedliche Klassen von ausführbaren Aktivitäten: Die so genannten Basisaktivitäten und die strukturierten Aktivitäten. Die vollständige Liste der **Basisaktivitäten** lautet:

- `<invoke>`
- `<receive>`
- `<reply>`
- `<assign>`
- `<throw>`
- `<terminate>`
- `<wait>`
- `<empty>`
- `<scope>`
- `<compensate>`

Die so genannten **strukturierten Aktivitäten** steuern den Fluss anderer Aktivitäten und können sowohl elementare als auch strukturierte Aktivitäten enthalten. Sie sind für die Ablaufsteuerung des Prozesses zuständig. Die vollständige Liste der strukturierten Aktivitäten lautet:

- `<flow>`
- `<sequence>`
- `<pick>`
- `<switch>`
- `<while>`

Die meisten Bezeichner der soeben vorgestellten Aktivitäten sprechen für sich selbst und werden an dieser Stelle nicht näher erläutert. Für eine übersichtliche Einführung in diese Aktivitäten wird auf das Buch **Service-orientierte Architekturen mit Web Services** [DJMZ05] verwiesen.
Eine Aktivität, die von zentraler Bedeutung für BPEL und für die späteren Ausführungen ist, ist die Basisaktivität `<invoke>`: Mittels `<invoke>` werden gemäß dem BPEL-Standard Aufrufe von externen Services durchgeführt. Die Syntax ist dabei etwas vereinfacht die Folgende:

Listing 3.3: Syntax des <invoke>-Elements

```
1 <invoke name="Service-Aufruf"
2   partner="externerService" portType="serviceNS:extService"
3   operation="methode1"
4   inputVariable="arg1" outputVariable="erg">
5   ...
6 </invoke>
```

Eine oberflächliche Betrachtung dieses Code-Fragments ergibt die Erkenntnis, dass die Methode `methode1` des Partner-Services `externerService` aufgerufen werden soll. Als Eingabe wird die Variable `arg1` vorgesehen und die Ergebnisse sollen in die Variable `erg` geschrieben werden. Die genaue Bedeutung der einzelnen Attribute ist für die späteren Ausführungen nicht von Bedeutung und wird daher hier nicht detaillierter ausgeführt.
Eine IBM-spezifische Erweiterung des `<invoke>`, die über den Standard hinausgeht, sind die so genannten Java-Snippets, die in Abschnitt 3.6 vorgestellt werden.

3.2.3 Prozess-Variablen

Geschäftsprozesse werden in BPEL, wie bereits dargestellt, mittels XML-Notation definiert. Variablen, die in Geschäftsprozessen verwendet werden, leiten sich daher von den in XML-Schema bekannten Datentypen ab. Die in XML verwendeten Datentypen fallen in zwei Klassen, welche im Folgenden charakterisiert werden:

Simple Datentypen Als simple Datentypen werden alle elementaren Datentypen wie Integer, Boolean, Float aber auch Date und String aufgefasst[2].

Komplexe Datentypen Als komplexe Datentypen werden Aggregationen mehrerer simpler oder auch komplexer Datentypen bezeichnet. Solche Datentypen werden im WebSphere Integration Developer auch als **Business Objects** bezeichnet. Sie können mit Hilfe eines gesonderten Business-Object-Editors in WID definiert werden.

3.2.3.1 Business Objects

Bei den so genannten Business Objects handelt es sich um Objekte komplexer Datentypen, wie sie im vorausgegangenen Abschnitt beschrieben wurden. Diese Datentypen werden in einer XML Schema Definition festgelegt, die an einen Namensraum gebunden wird. In den Geschäftsprozessen können diese Datentypen dann unter Angabe des entsprechenden Namensraumes verwendet werden.
Ein Beispiel für ein solches Business Object ist ein Datentyp wie `Kunde`, welcher in Abbildung 3.2 graphisch dargestellt ist. Die Abbildung stellt einen Screenshot aus dem Business-Object-Editor des WID dar, mit dessen Hilfe die XSD-Definitionen der Datentypen über eine graphische Oberfläche definiert werden können.

Wie aus der Abbildung zu ersehen ist, setzt sich der Datentyp Kunde aus 3 Komponenten zusammen: Vor- und Nachname des Kunden, sowie Adresse des Kunden[3]. Während die Komponenten Vor- und Nachname von dem simplen Typ `String` sind, handelt es sich beim Datentyp Adresse selbst wiederum um einen komplexen Datentyp.

[2] Eine Übersicht über alle simplen Datentypen, die XML Schema definiert kann auf `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#built-in-datatypes` (16.04.20007) eingesehen werden. Eine Auflistung sämtlicher simpler Datentypen bietet `http://www.w3.org/TR/xmlschema-0/#CreatDt` (16.04.2007)

[3] Die eckige Klammer hinter dem Datentyp Adresse bedeutet dabei, dass ein Kunde mehrere Adressen haben kann

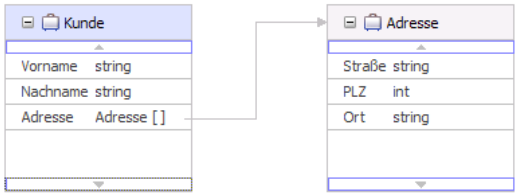

Abbildung 3.2: Beispiel für einen komplexen Datentyp 'Kunde'

Die Details, wie diese Datentypen in XML umgesetzt sind, sind unerheblich, da die Programmlogik auf diese Datentypen mittels einer speziellen API, der so genannten **Service Data Object**-API zugreift, welche im Folgenden beschrieben wird:

3.2.3.2 Service Data Objects

Die folgenden Ausführungen geben eine kurze Übersicht über das Konzept der Service Data Objects (SDO). Eine Einleitung in SDO im Java-Umfeld bietet die Arbeit **Introduction to Service Data Objects** [PB04], ausführlicher beschrieben wird die SDO-Technologie in dem Buch **Webanwendungen mit IBM Rational und IBM WebSphere V6** [GMR+06].

> „Zielsetzung von SDO ist es, ein einheitliches, universelles, objektorientiertes Framework für den Zugriff auf und die Bearbeitung von Daten zu bieten, welche aus unterschiedlichen Quellen stammen" [GMR+06].

Um diese Zielsetzung zu erfüllen, bietet SDO eine einheitliche API zum Zugriff auf Daten an, welche die Interna der Datenspeicherung vor dem Programmierer verbirgt. Für die Arbeit mit SDOs ist es völlig unerheblich, ob die Daten beispielsweise aus einer relationalen Datenbank oder einem XML-Dokument stammen. Über einen so genannten **Data Mediator Service** (DMS)[4] werden die Daten aus der jeweiligen Datenquelle geladen und Änderungen an den Daten dorthin zurückgeschrieben. Dieses Konzept ist in Abbildung 3.3 verdeutlicht und wird im Folgenden beschrieben:

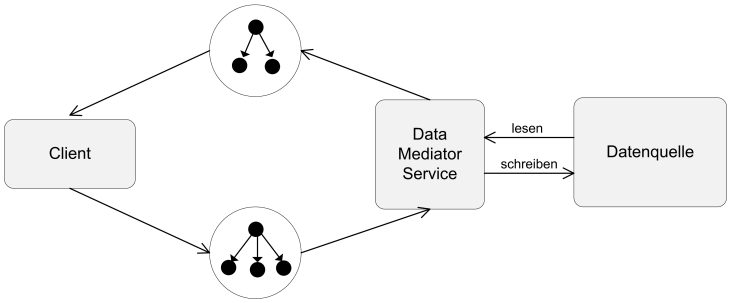

Abbildung 3.3: Zugriff auf Daten über einen Data Mediator Service

[4] Anmerkung: Diese Komponente wird auch als **Data Access Service** (DAS) bezeichnet.

Die SDO-Architektur basiert auf dem Konzept der *'disconnected data graphs'*[BBNP03]. Bei diesem Konzept erhält der Client einen Datengraph, also eine Datenstruktur, die baumartig organisiert ist, aus einer Datenquelle. Der Client arbeitet auf diesen Daten, indem er einzelne Knoten oder Blätter des Baumes abfragt oder überschreibt. Anschließend reicht er den veränderten Graphen zurück, woraufhin die Änderungen an dem Graphen auf der zugrunde liegenden Datenstruktur nachgeführt werden.
Der Zugriff auf die zugrunde liegenden Daten wird dabei durch den Data Mediator Service gekapselt, welcher sich um die Details der zugrunde liegenden Datendarstellung kümmert. Während der Arbeit auf dem Datengraphen ist der Client typischerweise nicht mit dem DMS verbunden, er verbindet sich lediglich um einen Datensatz abzufragen oder zu überschreiben. Dies ist der Grund für die Bezeichnung ***disconnected** data graphs*.

IBM hat eine eigene Implementierung des SDO-Konzepts vorgenommen. Die Arbeit mit Prozessvariablen unter Zuhilfenahme von IBMs SDO-Implementierung wird in Abschnitt 3.6.4 vorgestellt.

3.3 BPEL-Werkzeuge

Für die Entwicklung und Wartung von Geschäftsprozessen hat IBM die folgenden vier in Abbildung 3.4 dargestellten Phasen ausgemacht [WLP+06]:

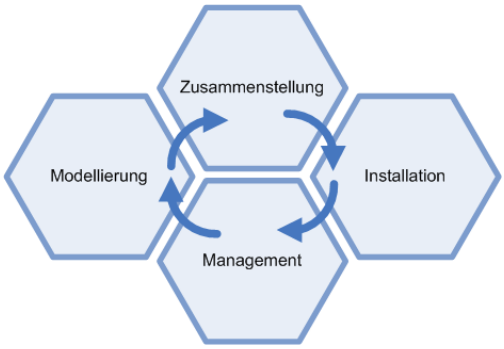

Abbildung 3.4: Entwicklungsphasen eines Geschäftsprozesses

- Die Phase der **Modellierung** umfasst die folgenden Aktivitäten: Erfassen, Simulieren, Analysieren und Optimieren von Geschäftsmodellen mit dem Ziel, das Geschäftsrisiko zu minimieren und gleichzeitig die Flexibilität zu erhöhen.
- Während der **Zusammenstellung** (englisch 'composition') wird die modellierte Lösung entwickelt, zusammengestellt und getestet.
- Auf diese Phase folgt die **Installation** (englisch 'deployment'), der Lösung auf einem Produktionsserver des Unternehmens.
- Während des Produktionseinsatzes sind die Komponenten einem permanenten **Management** unterworfen. Dieses Management überwacht und analysiert die Komponenten. Erkenntnisse aus dieser Phase können als Eingabe für ein konti-

nuierliches Re-engineering der Abläufe verwendet werden und stellen insofern die Grundlage für eine mögliche anschließende Modellierungsphase dar.

Für jede dieser vier Phasen stellt IBM ein dediziertes Werkzeug zur Verfügung, welches bei der Verwirklichung der Ziele in dieser Phase helfen soll. Die Zuordnung von Werkzeugen und Phasen ist in Tabelle 3.1 dargestellt.

Entwicklungsphase	**Werkzeug**
Modellierung	WebSphere Business Modeler
Zusammenstellung	WebSphere Integration Developer
Einsatz	WebSphere Process Server
Management	WebSphere Business Monitor

Tabelle 3.1: Zuordnung von BPEL-Werkzeugen zu Entwicklungsphasen

Für die Arbeit mit Geschäftsprozessen und BPEL gibt es zahlreiche Werkzeuge von einer Vielzahl von Herstellern. Besonders interessant sind dabei zwei Werkzeugklassen: Die so genannten BPEL-Engines – eine solche ist in WebSphere Process Server enthalten – und die BPEL-Editoren, zu denen WebSphere Integration Developer zählt. Die Eigenschaften dieser beiden Komponenten werden im Folgenden vorgestellt. Die Werkzeuge WebSphere Business Modeler, welcher in einer frühen Planungsphase eingesetzt wird, sowie WebSphere Business Monitor, welcher zur Optimierung von Geschäftsprozessen eingesetzt werden kann, spielen für diese Arbeit keine Rolle und werden nicht weiter vorgestellt. Eine umfassende Übersicht über sämtliche Komponenten gibt die Arbeit **Business Process Management**[WLP+06].

BPEL-Engine

Der zentrale Bestandteil der BPEL-Technologie ist die so genannte 'BPEL-Engine'. Die BPEL-Engine ist eine Laufzeitumgebung, die in der Lage ist, den mittels BPEL spezifizierten Geschäftsprozess zu interpretieren und bestimmungsgemäß auszuführen. Die BPEL-Engine kann beliebig realisiert werden; der BPEL-Standard schreibt in dieser Hinsicht nichts vor. Eine solche BPEL-Engine ist in IBMs WebSphere Process Server enthalten, welcher in Abschnitt 3.4 vorgestellt wird. Informationen über weitere BPEL-Engines sowie ein Vergleich von deren Leistungsfähigkeit können aus der Arbeit **Vergleich von BPEL Laufzeitumgebungen** [HRS06] entnommen werden.

BPEL-Editor

Eine weitere wichtige Komponente für die Arbeit mit BPEL-Prozessen sind spezielle BPEL-Editoren, die das Erstellen von BPEL-Prozessen erleichtern. Ein solches Werkzeug ist der WebSphere Integration Developer, welcher in Abschnitt3.5 vorgestellt wird. Eine gute Übersicht über die verfügbaren BPEL-Editoren sowie einen Vergleich von deren

Leistungsfähigkeit gibt die Arbeit **Vergleich von BPEL-Workflow Modellierungstools** [BKV05].

3.4 WebSphere Process Server

BPEL-Prozesse stellen eine formale Definition eines Geschäftsprozesses dar. Sie sind aber keine direkt ausführbaren Programme, sondern müssen interpretiert werden. IBMs WebSphere Process Server, dessen zentrale Eigenschaften im Folgenden beschrieben werden, ist in der Lage, BPEL-Prozesse auszuführen.

3.4.1 Architektur des WPS

WebSphere Process Server baut auf IBMs WebSphere Application Server (WAS) auf und erweitert diesen um eine Reihe von Funktionen. Die vereinfachte Architektur des WPS ist in Abbildung 3.5 dargestellt.

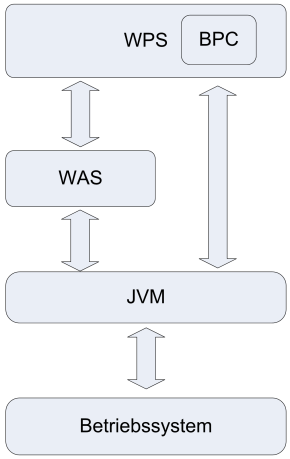

Abbildung 3.5: Architektur des WebSphere Process Server

Wie aus der Abbildung ersichtlich ist, laufen sowohl WPS als auch WAS auf einer Java Virtual Machine. Eine der zentralen Funktionen, die von WPS bereitgestellt wird, ist die Fähigkeit mittels BPEL definierte Prozesse ausführen zu können. Diese Funktionalität wird von einer Komponente des WPS mit Namen **Business Process Choreographer** (BPC) erbracht. Der BPC bildet damit den zentralen Bestandteil von IBMs BPEL-Technologie.
Sämtliche Sprachkonstrukte, die BPEL anbietet, müssen durch den BPC abgebildet werden. Dabei ist es interessant hervorzuheben, dass BPEL-Prozesse lediglich festlegen, **wann** etwas passieren soll. Zu der Art und Weise, **wie** dies umgesetzt werden kann macht der BPEL-Standard keinerlei Angaben.
Wie im Folgenden gezeigt wird, nutzt WPS diese Freiheit dazu, Laufzeitoptimierungen

an dem Geschäftsprozess vorzunehmen. Diese werden während der Installation des Prozesses vorgenommen.

3.4.2 Prozess-Installation

Der Standardweg, um einen Prozess auf WPS zu installieren – man spricht hier auch vom 'Process Deployment' – geht über ein Web-Interface des WPS. Über dieses Interface können mittels so genannter **Enterprise Application Archives** (EAR) vollständige Prozessdefinitionen auf WPS hochgeladen werden. Für die Installation der Geschäftsprozesse kann der Benutzer umfangreiche Einstellungen vornehmen, die in Form eines acht-seitigen Dialoges präsentiert werden.
Eine zweite Möglichkeit, die nur für Testzwecke verwendet werden kann, ist die Installation des Prozesses direkt aus der Entwicklungsumgebung WebSphere Integration Developer[5]. Für die Installation eines Prozesses genügt in diesem Fall ein einziger Knopfdruck auf die Schaltfläche 'publish': Damit werden alle notwendigen Schritte für die Installation auf der Testumgebung vorgenommen, wobei für die oben erwähnten Einstellungen Standardwerte verwendet werden. Details zum process deployment können der Hilfe des WebSphere Integration Developer entnommen werden [6].

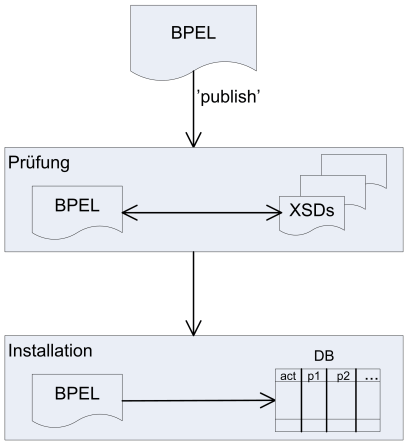

Abbildung 3.6: Übersicht der Schritte bei der Prozessinstallation

Wie aus Abbildung 3.6 ersichtlich ist, besteht die Installation eines Prozesses im Wesentlichen aus zwei Abschnitten: Der **Prüfung** auf Korrektheit und Vollständigkeit der Prozessdefinition sowie der eigentlichen **Installation** des Prozesses. Diese zwei Abschnitte werden im Folgenden beschrieben:

[5] Wie im Kapitel über den WID noch dargestellt wird, enthält WID zum Testen der Prozessdefinitionen einen vollständigen WPS als Testumgebung.

[6] Auch Online unter http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.runtime.doc/topics/tdeploy.html (zuletzt besucht 26.04.2007)

Prüfung

Zunächst wird die syntaktische Korrektheit des Prozesses sichergestellt. Zu diesem Zweck wird das BPEL-Dokument gemäß der XML Güte-Kriterien Wohlgeformtheit (wellformedness) und Gültigkeit (validity) überprüft. Kurz gesagt wird dazu jeder Teil des BPEL-Dokumentes auf die Einhaltung von Regeln, die in einer gesonderten Definition – der so genannten XML-Schema-Definition – festgehaltenen sind, überprüft. Mehr Details zum Thema Wohlgeformtheit und Gültigkeit werden in dem Abschnitt 6.1 vorgestellt oder können dem XML-Standard [BPSM+06] entnommen werden.
Gelingt die syntaktische Prüfung mittels der XSDs, so werden weitere semantische Überprüfungen durchgeführt, die mittels des Eclipse Modelling Frameworks (EMF) durchgeführt werden. Diese semantischen Prüfungen sind für diese Arbeit nicht relevant und werden daher nicht weiter beschrieben.

Installation

Wird die Korrektheitsprüfung erfolgreich durchlaufen, so folgt anschließend die eigentliche Installation des Prozesses. Dabei wird der Prozess analysiert und die einzelnen Aktivitäten des Prozesses extrahiert. Die auszuführenden Aktivitäten werden in einer speziellen Datenbank gespeichert, wobei zu jeder Aktivität neben den in der Prozessdefinition festgehaltenen Attributen zusätzlich gespeichert wird, welche Aktivität der vollständigen Ausführung dieser Aktivität folgen soll. Dieses Vorgehen ist in Abbildung 3.7 dargestellt[7].

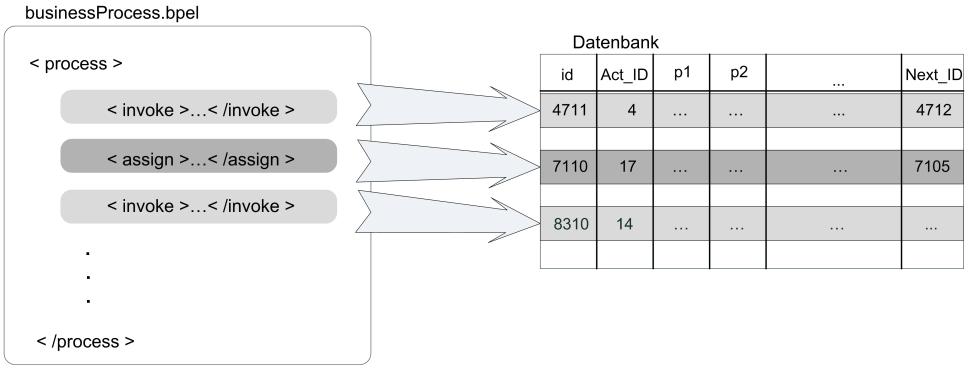

Abbildung 3.7: Installation eines Geschäftsprozesses

Auf diese Weise wird die Prozessdefinition von dem Informationsträger BPEL losgelöst und in eine wesentlich performantere Repräsentation übersetzt. Zur Laufzeit des Prozesses ist kein Zugriff auf die zugrunde liegende Prozessdefinition im BPEL-Format nötig.

[7] Bei der Abbildung handelt es sich um eine Verdeutlichung des zugrunde liegenden Konzeptes. Das Layout der Datenbank sowie die Bezeichnungen der einzelnen Felder sehen in Realität anders aus.

3.4.3 WPS-spezifische Erweiterungen zum BPEL-Standard

Der WebSphere Process Server definiert zwei herausragende Erweiterungen zum BPEL-Standard: Die so genannten Human Tasks sowie die Java-Snippets.
Kurz gesagt ermöglichen es die **Human Tasks**, Arbeitsschritte in BPEL-Prozesse zu integrieren, die von einem Menschen durchgeführt werden sollen. Eine genauere Betrachtung der Human Tasks ist für diese Arbeit nicht relevant, weitere Details können beispielsweise aus dem Whitepaper **WS-BPEL Extension for People - BPEL4People** [KKL+05] entnommen werden.
Die zweite BPEL-Erweiterung, die fester Bestandteil des WPS ist, sind die **Java-Snippets**. Diese werden in Abschnitt 3.6 vorgestellt. Für jegliche Erweiterungen, die in BPEL-Prozessen verwendet werden, gilt selbstverständlich, dass diesen Prozessen die Portabilität auf BPEL-Engines anderer Hersteller verloren geht. Dies ist eine wichtige Einschränkung und wird als solche hier besonders hervorgehoben.

Zusätzlich zu den soeben vorgestellten Erweiterungen bietet WPS die Möglichkeit der Erstellung eigener Erweiterungen zum BPEL-Standard. Von dieser Möglichkeit wird für die Umsetzung der Ziele dieser Arbeit in Abschnitt 6.2 Gebrauch gemacht.

3.5 WebSphere Integration Developer

Beim WebSphere Integration Developer (WID) handelt es sich um ein Eclipse-basiertes Werkzeug, mit dessen Hilfe BPEL-Geschäftsprozesse graphisch modelliert werden können. Wie schon der bereits vorgestellte WebSphere Process Server, so ist auch der WID ein sehr umfangreiches Werkzeug, dessen Funktionsumfang über das Modellieren von BPEL-Prozessen weit hinaus geht. An dieser Stelle wird jedoch hauptsächlich auf diesen Aspekt eingegangen.
Im Gegensatz zur BPEL-Engine, die eine unabdingbare Voraussetzung für den Umgang mit BPEL-Prozessen darstellt, ist ein graphischer BPEL-Editor – zumindest der Theorie nach – eine optionale Komponente: Theoretisch könnten BPEL-Prozesse allein mit einem Texteditor erstellt werden. Für die Praxis ist dieses Vorgehen allerdings nicht empfehlenswert: Die Umsetzung des verhältnismäßig einfachen Beispiels aus Abbildung 3.1 benötigt allein acht Seiten XML-Code zur Beschreibung. Mittels eines graphischen Editors kann die Erstellung von BPEL-Prozessen um Größenordnungen vereinfacht und beschleunigt werden. Abbildung 3.8 zeigt die graphische Benutzeroberfläche des WebSphere Integration Developer. Einige Bereiche von besonderem Interesse sind in dieser Grafik speziell hervorgehoben. Diese sind[8]:

Palette Die Palette (1) enthält Elemente, die per Drag&Drop in die Prozessdefinition aufgenommen werden können. Jedes der in BPEL (Version 1.1) verfügbaren Konstrukte wird durch eines dieser Symbole dargestellt. Zusätzlich enthält die Palette Symbole für die in der WebSphere-Familie üblichen BPEL-Erweiterungen, wie beispielsweise die Java-Snippets.

[8] Die verwendeten Bezeichnungen richten sich nach den Bezeichnungen, die in der WID-Hilfe verwendet werden, zu finden unter `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.bpel.ui.doc/concepts/cwbpel.html` (Zuletzt besucht 24.04.07)

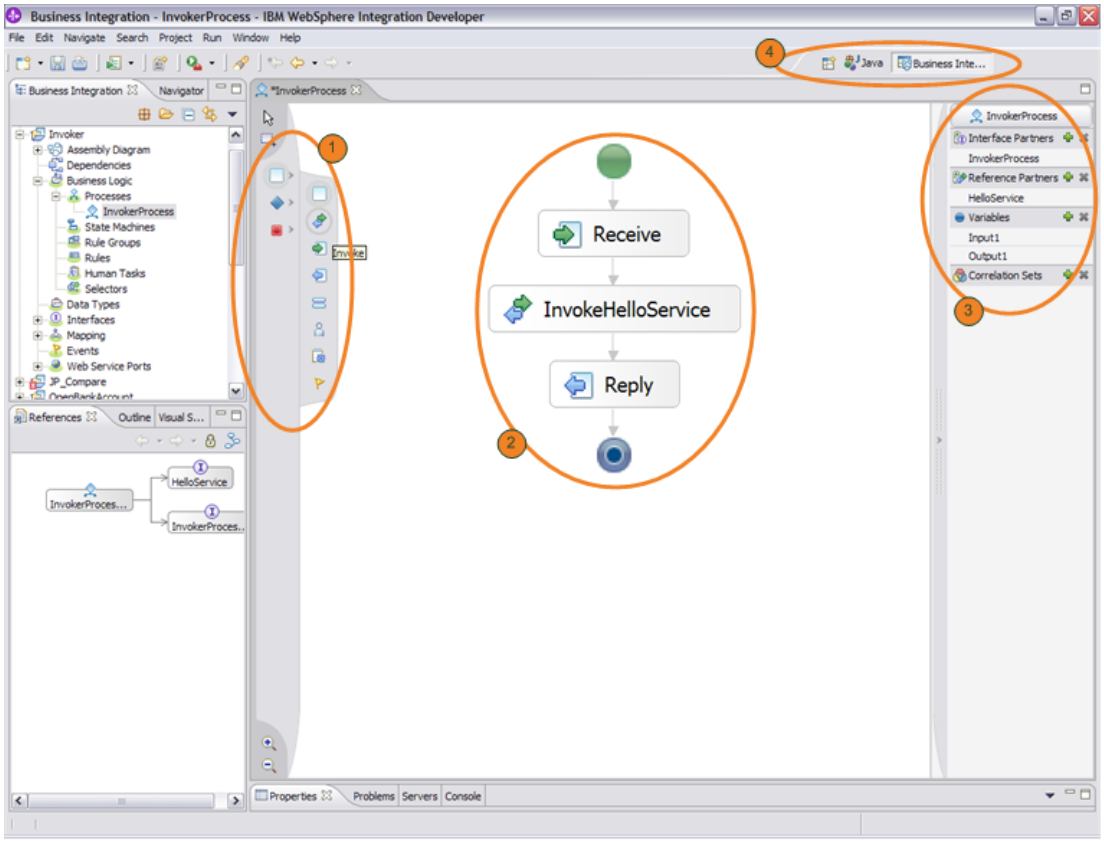

Abbildung 3.8: Graphische Benutzeroberfläche des WebSphere Integration Developer

Canvas Der so genannte Canvas (2) ist der weiße Bereich in der Mitte des Bildes. Die umrahmte Prozessdefinition enthält, abgesehen von einem Receive (Prozesseinstieg) und einem Reply (Prozessende) lediglich einen einzigen Service-Aufruf: Den Aufruf eines Hello-Services, in diesem Beispiel bezeichnet mit 'InvokeHello-Service'.

Tray In diesem Bereich (3) werden ausgewählte Informationen über den Prozess angezeigt, beispielsweise die im Prozess definierten Variablen. Mittels der Symbole im oberen rechten Rand der jeweiligen Kategorien können Objekte dieser Kategorie gelöscht, oder neue Objekte dieser Kategorie angelegt werden.

Perspektiven-Wahl Im oberen rechten Teil des WID (4) befindet sich eine Schaltflächensammlung, die den Wechsel der aktuellen Perspektive zulassen. Durch einen Perspektivenwechsel werden automatisch bestimmte Informationen eingeblendet und andere ausgeblendet. Auf diese Weise soll effizientes Arbeiten mit komplexen Szenarien ermöglicht werden.
Im aktuellen Beispiel ist gerade die **Business Integration**-Perspektive geöffnet, welche bei der Erstellung von BPEL-Prozessen verwendet wird.

3.5.1 Arbeiten mit Geschäftsprozessen

Zur Definition eines Geschäftsprozesses wählt der Benutzer aus einer vorgegebenen Palette von Elementen aus. Jedes von BPEL angebotene Konstrukt bekommt in WID eine graphische Repräsentation zugeordnet. Außerdem gibt es graphische Repräsentationen für die WPS-spezifischen BPEL-Erweiterungen (siehe 3.4). Während der Benutzer per Drag&Drop auf der graphischen Oberfläche die Eigenschaften des Geschäftsprozesses festlegt, wird im Hintergrund automatisch der entsprechende XML-Code erzeugt. Zu jedem Zeitpunkt ist es dabei möglich, die graphische Repräsentation des Prozesses zu schließen und den zugrunde liegenden XML-Code anzuzeigen. Dieser kann innerhalb von WID per Texteditor bearbeitet werden und anschließend wieder in eine graphische Repräsentation zurückverwandelt werden. Falls WID dabei auf Unstimmigkeiten trifft, so werden diese in der graphischen Repräsentation mit einem roten X gekennzeichnet.
Wie bereits im Abschnitt über WPS erwähnt wurde, lässt sich der mittels WID erstellte BPEL-Code direkt aus der graphischen Oberfläche heraus auf sein korrektes Verhalten hin überprüfen. Zu diesem Zweck enthält WID eine integrierte Testumgebung, die einen vollständigen WebSphere Prozess Server beinhaltet. Es ist außerdem möglich, die erstellten Prozessdefinitionen in ein spezielles '**Enterprise Application Archive**' (EAR) zu exportieren, welches auf einem externen WPS installiert werden kann.

3.5.2 Sichten auf einen Geschäftsprozesses

Zur Arbeit mit den Geschäftsprozessen bietet WID unterschiedliche Sichten (englisch 'views') an. Diese sind nicht zu verwechseln mit den bereits in Eclipse vorhandenen Perspektiven (englisch 'perspectives'). Mittels der **Eclipse-Perspektiven** lässt sich

der Arbeitskontext völlig austauschen. Mögliche Perspektiven sind beispielsweise die J2EE-Entwicklungs-Perspektive, die Plugin-Entwicklungs-Perspektive sowie die WID-spezifische 'business integration'-Perspektive, die sich auf die Entwicklung von Geschäftsprozessen bezieht.
Innerhalb der 'business integration'-Perspektive sind eine Reihe von **Sichten** auf gewisse Teilaspekte des Prozesses definiert. Beispielsweise können die Abhängigkeiten der verschiedener Komponenten in der 'Reference-view' eingesehen werden oder detaillierte Eigenschaften einer Komponente in der 'Properties-view'. Mehr Informationen dazu können der WebSphere Integration Developer Hilfe [IBM06a] entnommen werden [9].

3.5.3 Abstraktionsniveau

Die Entwicklung von Geschäftsprozessen geschieht – wie es für professionelle Softwareentwicklung üblich ist – auf zwei unterschiedlichen Abstraktionsleveln: Dem „Programmieren im Großen" und dem „Programmieren im Kleinen".
Das **Programmieren im Kleinen** bezieht sich im Fall von BPEL auf die Entwicklung der von einem Geschäftsprozess verwendeten Services. Diese können beliebig implementiert werden, solange sie per WSDL-Schnittstelle beschrieben und angesprochen werden können. Zur Entwicklung dieser Services sind detaillierte Angaben über die Funktionsweise des Services zu machen, meist wird hierfür eine Programmiersprache wie Java verwendet.
Das **Programmieren im Großen** dagegen abstrahiert über die Details der zugrunde liegenden Implementierung: Es sind lediglich Services und Methoden dieser Services bekannt. Durch Verknüpfen von Service-Aufrufen werden Lösungen für komplexere Probleme bereitgestellt. Auf dieser Ebene ist BPEL sowie der BPEL-Editor des WebSphere Integration Developer[10] angesiedelt.

Vorteile

Interessant an dieser Unterscheidung ist die Tatsache, dass mittels BPEL und entsprechenden Entwicklungswerkzeugen wie WID eine vollständige Trennung der Zuständigkeiten vorgenommen werden kann: Die IT-Abteilung kümmert sich um die Entwicklung von Web Services gemäß den Anforderungen des Unternehmens. Die Geschäftsprozesse eines Unternehmens können anschließend von Nicht-Informatikern erstellt und gepflegt werden: Es sind keine Kenntnisse einer bestimmten Programmiersprache oder spezieller Algorithmen nötig. Das in Abbildung 3.1 gezeigte Flussdia-

[9] Informationen zum Thema 'Business Integration perspective and views' liegen unter http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.6012.ui.doc/topics/rbiview.html

[10] Natürlich unterstützt WID aufgrund seines Eclipse-basierten Aufbaus auch die Entwicklung von Web Services und somit also das Programmieren im Kleinen. Gemeint ist an dieser Stelle aber die Geschäftsprozess-Komponente des WID

gramm kann – entsprechende Service-Komponenten vorausgesetzt – innerhalb weniger Minuten in einen lauffähigen BPEL-Geschäftsprozess umgesetzt werden.
Dies ermöglicht eine strikte Trennung der Zuständigkeiten und kann somit helfen, die Produktivität einzelner Arbeitsbereiche oder Mitarbeiter zu erhöhen. Die Tatsache, dass für die Zusammenstellung von Prozessen keine Programmiersprachenkenntnisse benötigt werden, soll hier noch einmal betont werden, da sie einen zentralen Baustein für die Motivation dieser Arbeit darstellt.

3.6 Java Snippet Activities

Wie bereits angesprochen, beschäftigt sich BPEL mit der Orchestrierung von Web Services. Die Tatsache, dass sich der BPEL-Standard im Wesentlichen auf diesen Aspekt beschränkt führt dazu, dass viele Hersteller von BPEL-Werkzeugen nicht-standardkonforme Erweiterungen zu BPEL definieren, um bestimmte Lücken zu füllen. Eine WebSphere-spezifische Erweiterung, die so genannten **'Java Snippet Activities'** werden im Folgenden vorgestellt.

3.6.1 Motivation für die Java Snippet Activities

Etwas zugespitzt formuliert, definiert BPEL lediglich, welche Services wann aufgerufen werden sollen. Zusätzlich dazu ist BPEL in der Lage einen internen Prozesszustand zu halten und die Prozessdaten geeignet zwischen den einzelnen Prozessaufrufen zu transportieren und gegebenenfalls zu transformieren.
Dieser klar beschränkte Funktionsumfang von BPEL birgt einen Nachteil: Für alles, was über die elementaren Funktionen von BPEL hinausgeht, muss ein Serviceaufruf durchgeführt werden. Selbst verhältnismäßig elementare Funktionen wie die Ausgabe von Zustandsinformationen auf die Konsole oder in eine Datei ist mit Standard-BPEL nicht möglich.

Aus diesem Grund hat IBM die so genannten 'Java Snippet Activities' eingeführt, welche auch verkürzt als Java-Snippets bezeichnet werden. Java-Snippets sind meist relativ kurze und einfach gehaltene Java-Code-Fragmente, die direkt in die BPEL-Dokumente integriert werden. Wie dies geschieht, wird in diesem Abschnitt gezeigt.

3.6.2 Eigenschaften der Java-Snippets

Innerhalb der Java-Snippets kann über spezielle API-Aufrufe auf Kontextinformationen des Prozesses wie beispielsweise dessen Namen zugegriffen werden oder Abfragen den Prozessstatus betreffend durchgeführt werden[11]. Außerdem ist ein direkter Zugriff auf sämtliche im Prozess definierten Variablen möglich. Bei Java-Snippets sind grundsätzlich nur die Seiteneffekte – wie Drucken auf die Konsole, oder Modifikationen an

[11] Eine Übersicht über diese API kann der WebSphere Integration Developer Hilfe unter http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.bpel.ui.doc/concepts/cjvameth.html entommen werden.

Prozessvariablen etc. – interessant, sie haben keinen Rückgabewert, beziehungsweise der Rückgabewert wird ignoriert. Eine weitere wichtige Eigenschaft der Java-Snippets ist es, dass Fehler zur Laufzeit eines Snippets der Laufzeitumgebung über eine Java RuntimeException signalisiert werden. Die Laufzeitumgebung des Prozesses reagiert auf diesen Ausnahmezustand standardmäßig mit dem Abbruch der Prozessausführung sowie einem vollständigen Rollback sämtlicher bereits durchgeführter Aktionen.
Für den Sprachumfang, der in Java-Snippets verwendet werden kann, macht IBM keine Vorschriften. Allerdings muss der Code J2EE-konform sein, darf also beispielsweise keine eigenen Threads starten.
Mit den Java-Snippets wird BPEL also um eine wichtige Komponente erweitert, die beispielsweise für das Debugging von Prozessen, aber auch für anspruchsvollere Aufgaben, verwendet werden kann. Der große Vorteil der Java-Snippets ist es, dass kleinere Aufgaben, die nicht von BPEL geleistet werden können, direkt im Prozess erledigt werden können ohne Aufruf eines externen Services. Dies kann die Ausführungsgeschwindigkeit des Prozesses deutlich erhöhen und verbessert zudem die Flexibilität der Prozessdefinition. Die wichtigsten Eigenschaften der Java-Snippets sind der Übersicht halber noch einmal in Tabelle 3.2 festgehalten.

Eingabe	keine
Rückgabe	keine
Funktionsumfang	Standard-J2EE-Sprachumfang + Zugriff auf Prozessvariablen + Zugriff auf Prozesskontext (spezielle API)
Fehlerfall	wird signalisiert über eine Java RuntimeException, führt zu Abbruch des Prozesses und Rollback

Tabelle 3.2: Übersicht über die Eigenschaften der Java-Snippets

3.6.3 Integration in BPEL

Die Java-Snippets wurden in BPEL – etwas vereinfacht dargestellt – auf die folgende Weise integriert:

Listing 3.4: Integration der Java-Snippets in BPEL

```
1 <invoke name="Beispiel-Snippet" operation="null" partnerLink="null" portType="null" ...>
2   <script>
3     <javaCode>
4       <![CDATA[ // Code der Java Snippet Activity ]]>
5     </javaCode>
6   </script>
7 </invoke>
```

Wie aus dem BPEL-Code ersichtlich ist, stellt der Container für das Java-Snippet eine Spezialform des in Abschnitt 3.2.2 vorgestellten `<invoke>` dar. Die Unterschiede zwischen dem in Abschnitt 3.2.2 vorgestellten `<invoke>` und dem hier vorgestellten `<invoke>` sind im Wesentlichen die Folgenden:

- Es handelt sich hier nicht um den Aufruf eines externen Services, daher können sämtliche Attribute des `<invoke>` mit 'null' belegt werden (Zeile 1).
- Das Element `<invoke>` enthält ein spezielles Kindelement mit dem Namen `<script>` (Zeile 2), welches in dem Unterelement `<javaCode>` den Java-Code des Snippets enthält.

3.6.4 Arbeiten mit den Prozessvariablen

Wie bereits in Abschnitt 3.2.3 vorgestellt, können Variablen in einem Geschäftsprozess unterschiedlich getypt sein: Sie können entweder von einem simplen oder einem komplexen Datentyp sein. In den Java-Snippets wird der Zugriff auf diese Variablen folgendermaßen umgesetzt:

Arbeiten mit Variablen mit simplem Datentyp

In Java-Snippets können Variablen mit einem simplen Typ ohne zusätzlichen Aufwand direkt wie lokale Variablen verwendet werden. Der Zugriff auf diese Variablen unterscheidet sich in den Java-Snippets in keiner Weise von dem für Java-Variablen üblichen Mechanismen. Damit dies funktioniert, muss ein Mapping der im Prozess definierten Variablen auf Java-Variablen durchgeführt werden. Konkret bedeutet dies ein Mapping von XML-Schema-Simple-Types auf Java-Typen. Dieses Mapping ist in der Hilfe zu WPS [IBM06b] beschrieben[12]. Die Arbeit mit einem simplen Datentyp innerhalb eines Java-Snippets wird in Listing 3.5 dargestellt.

Listing 3.5: Zugriff auf eine Variable mit simplem Typ in einem Java-Snippet

```
1  //lesender Zugriff auf eine Variable
2  System.out.println(simpleString);
3  //schreibender Zugriff auf eine Variable
4  simpleString = "neuer Wert";
```

Wie aus dem Listing ersichtlich ist, unterscheidet sich der Zugriff auf simpel getypte Variablen des Geschäftsprozesses in den Java-Snippets in keiner Weise von dem Zugriff auf normale Java-Variablen.

[12] Online unter http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.pmc.doc/ref/rjy1113.html, zuletzt besucht am 12.04.2007

Arbeiten mit Variablen mit komplexem Datentyp

Auf die Variablen mit komplexem Datentyp wird in den Java-Snippets über die in Abschnitt 3.2.3.2 vorgestellte SDO-API zugegriffen. Ein Beispiel für den Zugriff auf den in Abbildung 3.2 dargestellten Datentyp `Kunde` ist in Listing 3.6 abgedruckt.

Listing 3.6: Zugriff auf eine Variable mit komplexem Typ in einem Java-Snippet

```
1 //lesender Zugriff auf eine Variable
2 System.out.println(kunde1.getString("Vorname") + " " + kunde1.getString("Nachname"));
3 //schreibender Zugriff auf eine Variable
4 kunde1.setString("Vorname", "Max");
5 kunde1.setString("Nachname", "Mustermann");
```

Auch auf Variablen mit komplexem Datentyp kann ohne spezielle Ladevorgänge oder Ähnliches direkt zugegriffen werden. Allerdings unterscheidet sich der Zugriff auf diese Variablen geringfügig von dem Zugriff auf simpel getypte Variablen. Es werden spezielle `get`- und `set`-Methoden für das Lesen und Schreiben der einzelnen Felder verwendet. Beispielsweise wird in Zeile 5 das Feld mit dem Bezeichner `Nachname` des Objektes `kunde1` mittels der Funktion `setString()` auf einen neuen Wert gesetzt.

4 Motivation

In diesem Kapitel wird die Motivation für diese Arbeit ausführlicher dargestellt. Des Weiteren werden einige zentrale Eigenschaften der Skriptsprache PHP vorgestellt.

4.1 Problemstellung

Die Verwendung von Java-Snippets in BPEL-Prozessen bringt die im vorausgegangen Abschnitt dargestellten Vorteile. Die Verwendung von Java-Snippets birgt aber auch Probleme, denn es werden gleich zwei vorteilhafte Eigenschaften von BPEL damit ausgehebelt:

- Die **Portabilität** der Prozesse geht verloren: Prozesse, die Java Snippet Activities enthalten, sind nicht mehr unabhängig von den verwendeten BPEL-Werkzeugen. Java Snippets werden in dieser Form nur von der WebSphere-Produktfamilie unterstützt. Prozesse, die Java-Snippets enthalten, sind nicht ohne weiteres kompatibel zu Produkten anderer Hersteller.
- **Programmiersprachenkenntnisse**: Der in Abschnitt 3.5 über WebSphere Integration Developer besonders hervorgehobene Vorteil, dass für die Erstellung von BPEL-Prozessen keine Programmiersprachenkenntnisse nötig sind, geht verloren – zumindest, wenn der Funktionsumfang des WPS vollständig genutzt werden soll.

Zur Portabilität
Der erste Punkt in dieser Liste liegt in der Natur der Sache: Eine proprietäre Erweiterung ist nicht standardkonform. Da jedoch der von WID erzeugte BPEL-Code per Texteditor einsehbar ist, wird nicht ausgeschlossen, dass andere Hersteller kompatible Werkzeuge erzeugen. Zusätzlich besteht die Möglichkeit, diese Erweiterung oder ein

äquivalentes Konstrukt in einen zukünftigen BPEL-Standard mit einfließen zu lassen. Weitere offensichtliche Lösungsmöglichkeiten für dieses Problem gibt es nicht.

Zu den Programmiersprachenkenntnissen
Aus Anwendersicht mindestens genauso problematisch ist die Tatsache, dass mit Einführung der Java-Snippets nun Kenntnisse der Programmiersprache Java nötig sind, zumindest falls man das volle Potential von WebSphere Process Server ausschöpfen will. In Abschnitt 3.5 über WebSphere Integration Developer wurden bestimmte Benutzergruppen des WID jedoch explizit als Nicht-Informatiker ausgewiesen. Es entsteht ein offensichtlicher Konflikt zwischen der Mächtigkeit des Werkzeugs auf der einen Seite und der Qualifizierung der Benutzer des Werkzeugs auf der anderen Seite.
Dieser Zielkonflikt ist IBM bekannt und es wurde bereits versucht, dieses Problem zu beheben. Die von IBM bereits in den WID eingebaute Lösung des Problem ist der so genannte 'Visual Snippet Editor', der im Folgenden kurz vorgestellt wird.

4.2 Lösungsansatz: Visual Snippet Editor

Beim Visual Snippet Editor handelt es sich um ein graphisches Werkzeug, mit dessen Hilfe Java Code erzeugt werden kann. Der Benutzer wählt dabei aus den graphischen Repräsentationen einer Auswahl von Sprachkonstrukten. Im Hintergrund wird die graphische Repräsentation der Programmlogik in Java-Code übersetzt und im BPEL-Prozess gespeichert. Mehr Informationen über den Visual Snippet Editor kann der WID Hilfe [IBM06a] entnommen werden[1]. Abbildung 4.1 zeigt einen Ausschnitt der Benutzeroberfläche des Visual Snippet Editors.

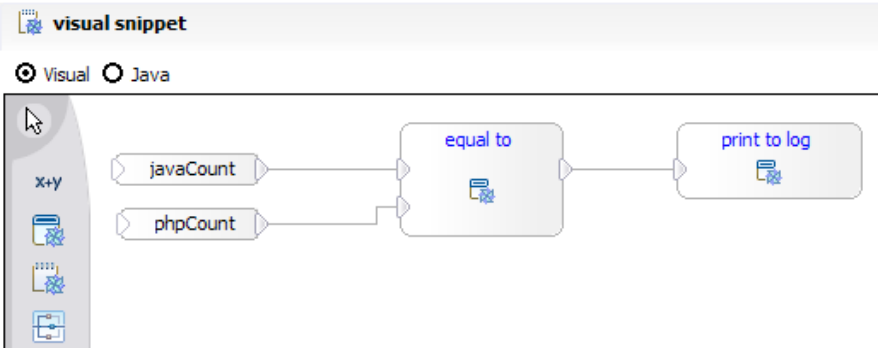

Abbildung 4.1: Benutzeroberfläche des Visual Snippet Editor

Die in der Abbildung gezeigte Programmlogik führt eine einfache Überprüfung auf Gleichheit der angegebenen Variablen durch und gibt das Ergebnis der Überprüfung aus. Am linken Rand der Abbildung ist ein Teil der Werkzeugleiste des Editors sichtbar.

[1]Der Abschnitt über den Visual Snippet Editor findet sich Online unter `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.activity.ui.doc/topics/cactund.html` (Zuletzt besucht am 16.04.2007)

Der durch den Visual Snippet Editor erzeugte Java-Code ist aus Listing 4.1 zu entnehmen:

Listing 4.1: Durch den Visual Snippet Editor erzeugter Java-Code

```
1 boolean __result__3;
2 {// equal to
3   __result__3 = (javaCount.intValue()) == (phpCount.intValue());
4 }
5 {// print to log
6   System.out.println((new java.lang.Boolean(__result__3)));
7 }
```

Wie aus dem Listing ersichtlich ist, ist die Java-Logik, die hinter dem Visual Snippet steckt sehr simpel, jedoch sind theoretisch auch komplexere Anwendungsszenarien denkbar.

Mittels des Visual Snippet Editor kann somit der oben angesprochene Zielkonflikt zwischen Flexibilität der Prozessdefintion und der Qualifikation der Benutzer, die mit dem Werkzeug arbeiten, behoben werden. Das Problem an dem Editor ist jedoch, dass die Einarbeitungsphase sowie die tägliche Arbeit mit dem Editor eine zeitintensive Aufgabe darstellt und dass die Bedienung im Allgemeinen als umständlich bezeichnet werden kann. Abgesehen davon erzeugt der Editor unübersichtlichen und suboptimalen Code.

4.3 Lösungsansatz: Andere Programmiersprachen

Die Lösung, die der Visual Snippet Editor bereitstellt, liegt darin, von der Programmiersprache zu abstrahieren und die Programmierung hinter einer Grafiklösung zu verbergen: Durch die graphische Repräsentation der Programmlogik wird formal ausgedrückt, was geschehen soll. Die Tatsache, dass dieser Formalismus im Hintergrund in Java-Code übersetzt wird, ist für den Benutzer unerheblich.
Eine alternative Lösung des Problems besteht dagegen darin, die formale Definition der Programmlogik nicht auf Java zu beschränken, sondern eine größere Zahl von Programmiersprachen zur Verfügung zu stellen, mit deren Hilfe formal ausgedrückt werden kann, was geschehen soll. Die Idee dahinter ist es, dass so die Verwendung des Visual Snippet Editors unnötig wird, da dem Benutzer somit vielleicht eine Programmiersprache geboten wird, die er bereits beherrscht.

Mögliche zusätzliche Programmiersprachen, die in Snippet-Aktivitäten angeboten werden können, fallen in zwei Klassen: Compilierte und interpretierte Sprachen. Dazu ist Folgendes anzumerken:

Compilierte Sprachen bieten gegenüber den interpretierten Sprachen den Vorteil, dass sie direkt auf der Maschine lauffähig sind und damit einen Geschwindigkeitsvorteil gegenüber den interpretierten Sprachen bieten. Der Nachteil für die Einbettung in BPEL liegt aber genau in dieser Tatsache: Snippet-Aktivitäten, die in einer compilierten Sprache formuliert sind, müssen vor ihrer Ausführung

compiliert werden. Diese Compilation kann entweder während der Installation des Prozesses vorgenommen werden – mit dem Nachteil, dass das entstehende Compilat in der Datenbank gespeichert werden muss – oder die Compilation geschieht zur Laufzeit des Prozesses, was sich negativ auf die Ausführungsgeschwindigkeit des Prozesses auswirkt. Beide Möglichkeiten sind unattraktiv und schließen somit die Verwendung von compilierten Sprachen aus.

Interpretierte Sprachen haben demgegenüber den klaren Vorteil, dass sie nicht compiliert werden müssen. Somit können die soeben beschriebenen Probleme mit compilierten Sprachen vermieden werden. Ein Vorteil von interpretierten Sprachen ist es außerdem, dass es möglich ist, die Interpretation des Snippets im selben Prozess auszuführen, in dem der Geschäftsprozess ausgeführt wird.

Aus diesem oberflächlichen Vergleich geht hervor, dass die interpretierten Sprachen für die Integration in BPEL besser geeignet sind als compilierte Sprachen. Eine Teilmenge der interpretierten Sprachen– die so genannten Skriptsprachen – zeichnet sich durch weitere Vorteile aus, die im folgenden Abschnitt dargestellt werden[2].

4.4 Skriptsprachen

Laut Definition der Wikipedia[3] sind Skriptsprachen durch folgende Eigenschaften charakterisiert:

> „Skriptsprachen sind Programmiersprachen, die vor allem für kleine, überschaubare Programmieraufgaben gedacht sind. Sie verzichten oft auf bestimmte Sprachelemente, deren Nutzen erst bei der Bearbeitung größerer Projekte zum Tragen kommen“.

Für viele Skriptsprachen gelten außerdem die folgenden Eigenschaften:

1. dynamische Typen
2. implizit deklarierte Variablen
3. automatische Speicherverwaltung

Diese Eigenschaften machen Skriptsprachen zu der bevorzugten Programmiersprachen-Kategorie für Snippet-Aktivitäten, denn genau dies ist der Einsatzzweck der Snippets: Es geht um relativ einfache und kurze Programmschnipsel, die auch von unerfahrenen Programmierern erstellt werden können sollten.

Wahl einer Skriptsprache

Aus der Vielzahl von Skriptsprachen, die für die verschiedensten Zwecke verwendet werden, zeichnen sich einige dadurch aus, dass sie besonders vielfältig eingesetzt

[2] Es gibt für einige Skriptsprachen die Möglichkeit diese zu compilieren. Diese Möglichkeit liegt jedoch nicht im primären Einsatzgebiet von Skriptsprachen.

[3] Zu finden unter `http://de.wikipedia.org/wiki/Skriptsprache`, (zuletzt besucht am 02.04.2007)

werden können. Zu diesen besonders universell einsetzbaren Skriptsprachen gehören beispielsweise [4]:

- Java Script
- Perl
- PHP
- REXX
- Visual Basic Script

Aus diesen Programmiersprachen wurde PHP für die weitere Betrachtung sowie die prototypische Implementation ausgewählt. Die Gründe, die für PHP sprechen sind:

- Weite Verbreitung, sowie sehr lebhafte PHP-Community,
- viele bereits existierende Erweiterungen für die verschiedensten Zwecke,
- eine C-ähnliche Syntax,
- sowie die Möglichkeit, eigene Erweiterungen für PHP zu entwickeln.

Des weiteren gibt es noch einen weniger offensichtlichen Grund: Das Haupteinsatzgebiet von PHP liegt in der Erstellung von dynamisch generierten Webseiten sowie Web-Applikationen. Die Vermutung liegt nahe, dass viele der PHP-basierten Web-Applikationen eine Menge von unterschiedlich komplexen Geschäftsprozessen implementieren. Das Spektrum deckt einen Großteil des eCommerce ab; beispielsweise Online-Shopping oder Online-Reisebüros.
Die Wartung der in dem PHP-Code integrierten Geschäftslogik ist in diesem Fall vergleichsweise teuer: Die richtige Stelle für eventuelle Änderungen muss im Quellcode der PHP-Anwendung aufgefunden werden, wohingegen bei BPEL in den meisten Fällen eine Wartung über die graphische Benutzeroberfläche des WID möglich ist.

Es ist daher denkbar, dass einige e-Commerce-Unternehmen, die von der zunehmenden Komplexität und den Wartungskosten ihrer im Quellcode der Web-Applikation verstreuten Geschäftslogik überfordert werden, den Wechsel auf hochspezialisierte Geschäftsprozess-Produkte wie beispielsweise die WebSphere-Familie von IBM planen. Für diese Unternehmen stellen die PHP-basierten Snippet-Aktivitäten des WPS möglicherweise einen Kaufanreiz gegenüber alternativen BPEL-basierten Geschäftsprozess-Werkzeugen dar. Auf diesen Aspekt wird in Kapitel 7 etwas detaillierter eingegangen.

4.5 PHP

PHP wurde ursprünglich von Rasmus Lerdorf für webbasierte Einsatzzwecke entwickelt und auch heute ist dies noch das Haupteinsatzgebiet von PHP. Jedoch wurde das Spektrum der Einsatzmöglichkeiten von PHP kontinuierlich verbreitert. Mittels einer großen Vielfalt von Erweiterungen ist PHP mittlerweile für die unterschiedlichsten Anforderungen gerüstet.

[4] laut Wikipedia: http://de.wikipedia.org/wiki/Skriptsprache (zuletzt besucht am 02.04.2007)

Einige zentrale Eigenschaften von PHP sind nach Gaylord Aulke [Aul05] die Folgenden:

- Hochoptimiert für Web-Applikationen und WebServices.
- Oft als Verbindungssprache (**glue language**) angesehen. Mittels PHP kann eine leichtgewichtige Integration verschiedener open source Bibliotheken sowie C-APIs propieterer Softwareprodukte (Datenbanken, SAP-Systeme etc) vorgenommen werden.
- Benutzt dynamisch getypte Variablen.
- Enthält sehr umfangreiche Funktionen zum Umgang mit Strings und Arrays

Zur Beliebtheit von PHP tragen laut Joe MacCabe [McC07] die folgenden Eigenschaften von PHP bei:

- Die Einfachheit und
- die große Anzahl existierender PHP-Erweiterungen
- sowie eine umfangreiche Auswahl frei verfügbarer Web-Applikationen, wie beispielsweise Datenbankmanagement und Contentmanagement.

Erweiterbarkeit von PHP

Erweiterungen für PHP können von speziellen Webseiten heruntergeladen und auf einfache Weise in PHP Skripten geladen und verwendet werden. Die offiziellen Webseiten für PHP-Erweiterungen sind die **PHP Extension Community Library** (PECL)[5] und das **PHP Extension and Application Repository** (PEAR)[6], es ist jedoch auch eine Vielzahl von Erweiterungen für PHP über die Webseiten von Drittanbietern verfügbar.

Existierende PHP-Erweiterungen sind beispielsweise:

- Die **XML-Erweiterung** bietet umfangreiche Möglichkeiten zur Arbeit mit XML-Dokumenten
- Mittels der **SDO-Erweiterung** wird der Zugriff auf Datenstrukturen über das in Abschnitt 3.2.3.2 beschriebene SDO-Konzept ermöglicht.
- Über diverse Datenbankerweiterungen, beispielsweise die **MySQL-Erweiterung**, kann eine Vielzahl gängiger Datenbanken in PHP verwendet werden.

Einige PHP-Erweiterungen zählen standardmäßig zur PHP-Installation und sind aus Sicht des Benutzer vom Standard-Sprachumfang nicht zu unterscheiden. Ein Beispiel für eine solche Erweiterung ist die oben aufgelistete XML-Erweiterung.

Abgesehen von der Verwendung existierender PHP-Erweiterungen unterstützt PHP auch die Entwicklung eigener PHP-Erweiterungen. Im Folgenden werden die unterschiedlichen Möglichkeiten, PHP zu erweitern, miteinander verglichen.

[5] zu finden unter `http://pecl.php.net/`
[6] zu finden unter `http://pear.php.net/`

Vergleich der Erweiterungsmöglichkeiten

Erweiterungen für PHP fallen grundsätzlich in zwei unterschiedliche Klassen:

- PHP-Erweiterungen, die in der Sprache PHP geschrieben sind und
- PHP-Erweiterungen, die mit der Programmiersprache C erstellt wurden.

Erweiterungen, die in der Sprache PHP geschrieben sind, haben den Vorteil, dass sie verhältnismäßig einfach zu erstellen und zu warten sind: Es wird lediglich ein Texteditor und die Kenntnis der Programmiersprache PHP benötigt. Dagegen ist für Erweiterungen, die mit der Programmiersprache C erstellt werden mehr Aufwand nötig: Beispielsweise werden Kenntnisse der Programmiersprache C sowie einiger PHP-Interna und PHP-spezifischer API-Aufrufe benötigt.
Allerdings haben in C erstellte PHP-Erweiterungen den Vorteil, dass sie zur Laufzeit schneller sind als PHP-basierte Erweiterungen. Außerdem kann nur mit dieser Art der Erweiterung der PHP-Interpreter selbst um neue Funktionalität erweitert werden. Streng genommen handelt es sich bei den PHP-basierten Erweiterungen lediglich um Funktionsbibliotheken; der in PHP zur Verfügung stehende Funktionsumfang wird dadurch nicht erweitert. Dies ist nur mit einer C-basierten Erweiterung der PHP-Laufzeitumgebung möglich. In C geschriebene PHP-Erweiterungen müssen unter Microsoft Windows in Form einer DLL bereitgestellt werden. Wie diese geladen und verwendet werden können, wird im Folgenden beschrieben.

Laden einer PHP-Erweiterung

PHP-Erweiterungen können beim Starten des PHP-Interpreters oder zur Laufzeit eines Skriptes geladen werden. Zur Laufzeit des Skriptes kann eine PHP-Erweiterung mittels der PHP-Funktion `dl()` unter Angabe des Erweiterungsnamens geladen werden. Soll beispielsweise die Erweiterung für das Arbeiten mit Service Data Objects (kurz SDO) geladen werden, so kann dies über den Befehl `dl(php_sdo.dll)` in einem PHP-Skript geschehen.
Soll die Erweiterung bereits beim Start des PHP-Interpreters geladen werden, so kann dies in der Konfigurationsdatei des PHP-Interpreters (Standardname php.ini) angegeben werden. Dies geschieht über die `extension`-Direktive. Für das obige Beispiel muss der Eintrag in die Konfigurationsdatei des PHP-Interpreters folgendermaßen aussehen: `extension=php_sdo.dll`.

Nach dem Laden der Erweiterung können die in der Erweiterung definierten Funktionen in den PHP-Snippets auf die gleiche Weise verwendet werden, wie auch die Standard-PHP-Sprachkonstrukte. Die per Erweiterung zur Verfügung gestellte Funktionalität ist aus Sicht des PHP-Programmierers nicht zu unterscheiden von dem Sprachumfang, der vom Standard-PHP-Interpreter zur Verfügung gestellt wird.
In der Praxis ist es sogar so, dass einiges der Funktionalität, die in PHP standardmäßig zur Verfügung steht, in Wirklichkeit über Erweiterungen des PHP-Kerns realisiert ist.

4.6 Zusammenfassung

Durch die Einführung von zusätzlichen Sprachen für die Snippet-Aktivitäten kann die anfangs beschriebene Problematik, die aufgrund der Beschränkung auf die Programmiersprache Java besteht, verringert werden:
Es reicht danach, eine beliebige Programmiersprache aus einer größeren Anzahl unterstützter Sprachen zu beherrschen. Auf diese Weise kann der Umgang mit dem zeitraubenden Visual Snippet Editor unterbleiben und gleichzeitig die durch die Snippets eingeführte Flexibilität bewahrt werden.

Diese Arbeit analysiert, wie Skriptsprachen in BPEL eingebunden und zur Laufzeit ausgeführt werden können. Für diese Untersuchungen wird das Beispiel PHP und WebSphere Process Server verwendet.

5 Analyse

In diesem Kapitel werden die unterschiedlichen Möglichkeiten, wie der in den PHP-Snippets enthaltene PHP-Code ausgeführt werden kann, vorgestellt und bewertet. Diese Ausführungen beziehen sich auf das konkrete Beispiel PHP und WebSphere Process Server, sie können jedoch – zumindest teilweise – auf andere Skriptsprachen sowie BPEL-Engines übertragen werden.

5.1 Übersicht

Die Modifikationen, die im Einzelnen nötig sind, um PHP in den Snippet-Aktivitäten verwenden zu können, beziehen sich im Wesentlichen auf die folgenden, von einander unabhängigen, Entwicklungsschritte:

- Es muss eine **Erweiterung zum BPEL-Standard** definiert werden, die in der Lage ist, den PHP-Code für die Snippet-Aktivität zu transportieren. Diese wird in Abschnitt 6.1 vorgestellt.
- Für alle Sprachelemente, die über den Standard-BPEL-Sprachumfang hinausgehen, muss eine **Erweiterung für WPS** entwickelt werden, die das Folgende leistet:
  - Bei der Prozess-Installation sorgt die WPS-Erweiterung dafür, dass der Prozess als gültig erkannt wird und dass alle relevanten Daten – speziell der Programmcode des PHP-Snippets – in der Datenbank abgespeichert werden. Die Maßnahmen, die dazu nötig sind, werden in Abschnitt 6.2.1 vorgestellt.
  - Zur Laufzeit des Prozesses ermöglicht die Erweiterung die Ausführung des PHP-Codes durch den PHP-Interpreter (siehe Abschnitt 6.3) und ermöglicht gleichzeitig dem PHP-Snippet den Zugriff auf die Prozessvariablen (siehe Abschnitt 6.4).

Die Interaktion zwischen WPS und dem PHP-Interpreter, die für die Ausführung des PHP-Interpreters sowie den Zugriff auf die Prozessvariablen nötig ist, kann auf unterschiedliche Weisen realisiert werden. Das grundlegende Konzept für die Erweiterungen von WPS und PHP sowie die möglichen Technologien dafür werden, ausgehend von den in Abschnitt 5.2 festgehaltenen Anforderungen an die PHP-Snippets, in diesem Kapitel vorgestellt und bewertet.

5.2 Anforderungen

Für die Wahl des richtigen Konzepts und der richtigen Technologie sind die Anforderungen, die an die Auswertung von PHP-Snippets gestellt werden, sowie der in PHP-Snippets geforderte Funktionsumfang von zentraler Bedeutung. Sie geben den Kriterien- und Bewertungskatalog für die Tauglichkeit der vorgestellten Technologien vor. Die Anforderungen an die Implementierung der PHP-Erweiterung sind im Wesentlichen die Folgenden:

1. PHP-Programmierer sollten alle aus PHP gewohnten Konstrukte verwenden können. Die Programmierung eines PHP-Snippets sollte sich nicht wesentlich von der normalen PHP-Programmierung unterscheiden.
2. Aus funktionaler Sicht sollten sich die PHP-Snippets nicht von den Java-Snippets unterscheiden, das heißt der gesamte von den Java-Snippets gebotene Funktionsumfang sollte in PHP auch zur Verfügung stehen. Insbesondere sollte in den PHP-Snippets der lesende und schreibende Zugriff auf die im Prozess definierten Variablen (Prozessvariablen) möglich sein.
3. Die Sicherheit des Systems sollte durch Einführung von PHP-Snippets nicht vermindert werden. Auch sollte die Performanz des Systems soweit als möglich erhalten bleiben.

5.2.1 Bewertungskatalog

Aus den soeben genannten Anforderungen lassen sich die folgenden konkreten Eigenschaften für die einzelnen Umsetzungsmöglichkeiten ableiten. Die aufgeführten Eigenschaften dienen später als Bewertungsgrundlage für die Tauglichkeit der vorgestellten Möglichkeiten und stellen somit deren Bewertungskatalog dar:

- **Kommunikation**: Zwischen dem PHP-Snippet-Code und dem BPEL-Prozess-Kontext muss eine bidirektionale Kommunikation mit Direktzugriff möglich sein. Der Grund dafür wird direkt im Anschluss an diesen Bewertungskatalog vorgestellt.
- **Ausnahme-Behandlung**: Wie in Abschnitt 3.6 über die Java-Snippets dargestellt wurde, werden Fehler, die zur Laufzeit eines Java-Snippets auftreten, mittels einer Java RuntimeException signalisiert. Dieser Mechanismus sollte ebenfalls für die Ausführung von PHP-Snippets gelten. Wie in den folgenden Ausführungen

noch gezeigt wird, kann dies auf unterschiedlich leistungsfähige Art und Weise geschehen.

- **Systemsicherheit**: Durch die Ausführung von PHP-Code sollen keine Sicherheitslücken im System entstehen.
- **Ausfallsicherheit**: Die Integration von PHP darf nicht dazu führen, dass die Zuverlässigkeit und Erreichbarkeit des Gesamtsystems durch die PHP-Komponente beeinträchtigt wird.
- **Performanz**: Die Auswertung des PHP-Codes sollte so schnell und kostengünstig wie möglich durchgeführt werden.

Zusätzlich zu den oben aufgeführten Punkten gilt außerdem die folgende Forderung:

- **Sprachumfang**: Der Sprachumfang, den PHP bietet, soll in vollem Umfang in den PHP-Snippets zur Verfügung stehen.
  Diese Forderung ist keine Selbstverständlichkeit, was am architekturellen Aufbau von PHP liegt: Der Standard-PHP-Sprachumfang kann sehr einfach durch PHP-Erweiterungen, wie sie beispielsweise durch die PHP-Erweiterungsmechanismen PECL[1] oder PEAR[2] bereitgestellt werden, ergänzt werden.

Aufgrund der Wichtigkeit der letzten Forderungen wurden Lösungen, bei denen zu erwarten ist, dass sie den Sprachumfang nicht vollständig abdecken, von vorne herein aus dem Vergleich ausgeschlossen. Ein Beispiel für eine solche Technologie ist die Interpretation des PHP-Codes durch einen Java-basierten PHP-Interpreter. Dieser kann den Zugriff auf PHP-Erweiterungen, wenn überhaupt, nur auf Umwegen anbieten.
Alle Technologien, die im Folgenden vorgestellt werden, verwenden im Hintergrund den Standard-PHP-Interpreter, welcher die maximale Verfügbarkeit sämtlicher PHP-Sprachkonstrukte gewährleistet.

5.2.2 Bidirektionale Kommunikation mit Direktzugriff

Im Bewertungskatalog wird unter der Überschrift **Kommunikation** eine bidirektionale Kommunikation mit Direktzugriff zwischen der PHP-Ausführungskomponente und der Java-Laufzeitumgebung gefordert. Diese Eigenschaft wird im Folgenden motiviert:

Bidirektionale Kommunikation

In den Anforderung an die Umsetzung der PHP-Snippets findet sich unter Punkt zwei die Forderung, dass PHP Zugriff auf die Prozessvariablen haben muss. Des Weiteren wird gefordert, dass der Zugriff auf diese Ressourcen sowohl lesend als auch schreibend möglich sein muss. Wie in wie Abbildung 5.1 angedeutet ist, impliziert dies

[1] PECL ist die Abkürzung für 'The PHP Extension Community Library', eine Sammlung von PHP-Erweiterungen, die im Binärformat zur Verfügung gestellt werden und direkt auf der Maschine lauffähig sind. Diese werden Beispielsweise in Form einer .dll oder .so-Datei bereitgestellt.

[2] PEAR ist die Abkürzung für 'PHP Extension and Application Repository' welches ebenfalls einen Erweiterungsmechanismus für PHP bereitstellt. Im Gegensatz zu PEAR sind hier die Erweiterungen allerdings in der Sprache PHP geschrieben.

eine bidirektionale Kommunikation zwischen dem Prozesskontext auf der einen Seite und dem PHP-Snippet auf der anderen Seite:

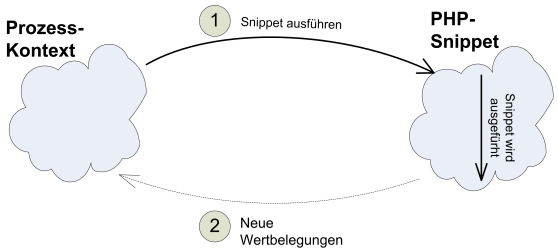

Abbildung 5.1: Bidirektionale Kommunikation

Nur mittels einer Kommunikation, die bidirektional ausgelegt ist, ist es möglich,

- einerseits die Ausführung des PHP-Snippets zu veranlassen, sowie dem Snippet Informationen zu übergeben (1),
- sowie andererseits die Informationen, die bei der Ausführung des Snippets verändert werden, zurück zu übermitteln (2).

Eine bidirektionale Kommunikation kann prinzipiell auf zwei unterschiedliche Weisen realisiert werden, nämlich mittels **'Copy-in Copy out'** oder **Direktzugriff**. Was damit gemeint ist, wird im Folgenden erläutert.

Copy-In Copy-Out

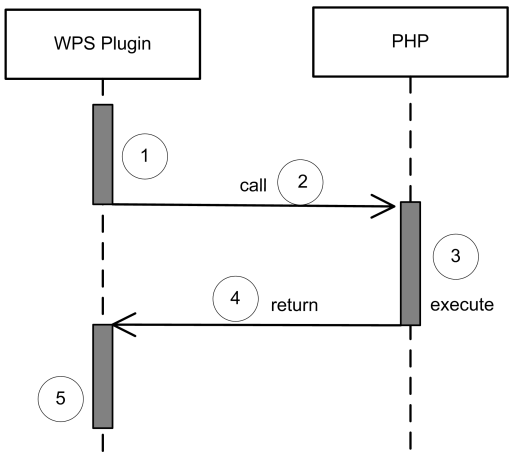

Abbildung 5.2: Copy-In Copy-Out-Kommunikationsschema

Während der Ausführung des Geschäftsprozesses (1) tritt die Notwendigkeit auf, ein PHP-Snippet auszuwerten. Zu diesem Zweck wird der PHP-Interpreter aufgerufen (2), wobei diesem zu Beginn der PHP-Ausführung erstens das auszuführende PHP-Snippet und zweitens pauschal sämtliche verfügbaren Geschäftsprozess-Informationen übergeben werden. Der PHP-Interpreter arbeitet mit diesen Informationen (3), wobei

er lesend und schreibend auf einer Kopie der Daten arbeitet. Am Ende der PHP-Ausführung werden sämtliche Informationen zurück übermittelt (4). Die von PHP durchgeführten Änderungen am Datensatz werden im Java-Kontext entsprechend nachgeführt (5), sofern nötig.
Dieses Kommunikationsverfahren kommt vor allem aus Gründen der **Effizienz** nicht in Frage: Die Variablen und Kontextinformationen können mitunter einen erheblichen Umfang erreichen. Es ist sehr wahrscheinlich, dass das Snippet in vielen Fällen nur auf einen Bruchteil der bereitgestellten Information wirklich zugreift. Eine pauschale Bereitstellung für jedes Skript kann somit einen untragbaren Overhead erzeugen.

Direktzugriff

Mittels einer bidirektionalen Kommunikation mit Direktzugriff können die oben genannten Probleme weitestgehend umgangen werden.

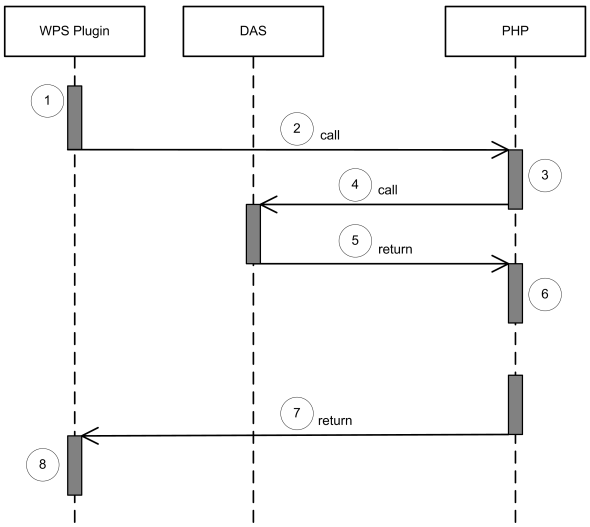

Abbildung 5.3: Kommunikation mittels Direktzugriff

Während der Ausführung eines Geschäftsprozesses (1) tritt die Notwendigkeit auf, ein PHP-Snippet auszuwerten. Dazu wird auch hier der PHP-Interpreter aufgerufen, allerdings wird diesem lediglich der Snippet-Code sowie einige wenige Informationen, die zum Zugriff auf die Prozessinformationen benötigt werden, übergeben (2). Während der Ausführung des PHP-Codes(3) tritt die Notwendigkeit auf, auf Informationen des Geschäftsprozesses zuzugreifen. Dazu greift der PHP-Interpreter über spezielle Hilfsfunktionen auf einen Data Access Service (DAS) zurück, der Zugriff auf die Prozessinformationen bereitstellt (4). Der DAS liefert diese Informationen an den PHP-Interpreter (5) und die Ausführung des PHP-Snippets wird wieder aufgenommen (6). Die Vorgänge (4) und (5) können anschließend beliebig häufig wiederholt werden, um Informationen im Prozesskontext zu lesen oder zu schreiben. Sobald das PHP-Snippet vollständig durchgelaufen ist, wird zum Aufrufer zurückgekehrt (7), wobei in diesem

Fall keinerlei Informationen zurück übermittelt werden. Der Geschäftsprozess läuft sofort weiter (8), ohne dass spezielle Aktionen zur Datennachführung durchgeführt werden müssen.

5.3 Mögliche Technologien

Um die Auswertung von PHP-Code zur Laufzeit des BPEL-Prozesses zu ermöglichen, können einige in ihrer Qualität völlig unterschiedliche Wege beschritten werden. Diese werden in diesem Abschnitt vorgestellt und bewertet.

5.3.1 Allgemeines

Für die Realisierung der PHP-Snippets ist grundsätzlich eine Erweiterung des WPS sowie eine Erweiterung der PHP-Laufzeitumgebung nötig und zwar aus den folgenden Gründen:

- Die Erweiterung der PHP-Laufzeitumgebung ist nötig, um den Zugriff auf die Prozessvariablen aus den PHP-Snippets heraus zu ermöglichen: Diese hochspezialisierte Funktionalität ist selbstverständlich nicht Bestandteil des PHP-Sprachumfangs.
- Die Erweiterung des WPS ist aus zwei Gründen nötig:
  - Zum einen wird eine Möglichkeit benötigt, den PHP-Interpreter anzusprechen, um die Ausführung von PHP-Code zu veranlassen.
  - Außerdem werden für den Zugriff auf die Prozessvariablen spezielle Callback-Funktionen benötigt, welche ebenfalls durch das Plugin bereitgestellt werden müssen.

Die für die Interaktion beider Komponenten nötige Kommunikation kann, wie bereits angesprochen, auf unterschiedliche Weisen realisiert werden, welche in diesem Kapitel verglichen werden.
Um den Vergleich der einzelnen Technologien möglichst übersichtlich zu gestalten, werden sämtliche vorgestellten Technologien nach einem einheitlichen Muster präsentiert. Die einzelnen Themenbereiche, die angesprochen werden, sind die Folgenden:

Realisierung In diesem Teil wird beschrieben, wie die Ausführung von PHP mittels dieser Technologie umgesetzt werden kann. Insbesondere werden die folgenden Fragen beantwortet:

- **Integration**: Wie wird die Ausführung von PHP-Code ermöglicht? Oder anders ausgedrückt: Auf welche Weise wird WPS erweitert, um die Ausführung von PHP-Code durchzuführen?
- **Zugriff auf Prozessvariablen**: Auf welche Weise wird der Zugriff auf die Prozessvariablen in PHP ermöglicht?

Bewertung Im Anschluß an die Beschreibung der Realisierung erfolgt eine Bewertung anhand der in Abschnitt 5.2.1 festgelegten Bewertungskriterien. Diese sind:

Kommunikation, Ausnahme-Behandlung, Ausfallsicherheit, Systemsicherheit und Performanz.

Symbolik

Die Ausführungen in diesem Kapitel stellen die verschiedenen Technologien nur oberflächlich vor, aus diesem Grund werden zur Verdeutlichung der unterschiedlichen Konzepte vereinfachte Graphiken verwendet, welche sich der folgenden Symbolik bedienen:

Ein Erweiterung einer Komponente mittels eines Plugins oder über einen Standard-Erweiterungsmechanismus wird durch das Symbol in Abbildung 5.4 dargestellt.

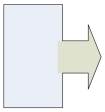

Abbildung 5.4: Symbol für eine Erweiterung einer Komponente

Bei der Interaktion zweier Komponenten tritt in einer konkreten Kommunikation immer eine der beiden Komponenten als Anbieter (Server) eines Services oder einer Funktion auf während die andere Komponente als Aufrufer (Client) in der Kommunikation auftritt. Dieser Sachverhalt wird mittels der Symbolik aus Abbildung 5.5 angedeutet, wobei innerhalb des Kreises der verwendete Kommunikationsmechanismus festgehalten wird. In dem hier verwendeten Beispiel bedeutet dies, dass die Kommunikation zwischen den beiden Parteien über SOAP abgewickelt wird.

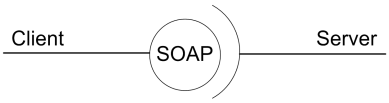

Abbildung 5.5: Symbol für eine Schnittstelle

5.3.2 Web Service

Im Kontext von BPEL und Web Services ist es eine nahe liegende Idee, die Auswertung der PHP-Snippets über einen externen Web Service zu realisieren. Diese Realisierungsmöglichkeit wird im Folgenden vorgestellt.

Realisierung

PHP kann mit Hilfe einer PHP-Erweiterung in die Lage versetzt werden, in Web Service Kommunikationen sowohl als Client als auch als Server aufzutreten. Dazu

gibt es eine Reihe von unterschiedlichen Erweiterungen für PHP. Die Details der einzelnen Erweiterungen sind an dieser Stelle irrelevant. Die folgenden Beschreibungen beziehen sich auf die 'PHP SOAP Extension'[Sto04] welche in der Programmiersprache C geschrieben ist und per DLL in den PHP-Interpreter geladen werden kann.

Integration von PHP über WSDL und SOAP
Mittels der SOAP-Erweiterung für PHP ist es möglich Web Services mit PHP zu realisieren. Die Schnittstelle zu den Services kann wie bei Web Services üblich als WSDL-Datei publiziert werden. Die Integration des PHP-Interpreters über diese Methode kann ohne große Umstände vorgenommen werden: Aufgrund der Tatsache, dass der PHP-Interpreter in diesem Setup als Web Service angeboten wird, kann eine Erweiterung von BPEL und von WPS unterbleiben: Wann immer ein PHP-Snippet ausgeführt werden soll, wird ein Aufruf des entsprechenden Web Services – über ein Standard-`<invoke>`– vorgenommen. Dieser Aufbau ist in Abbildung 5.6 dargestellt.

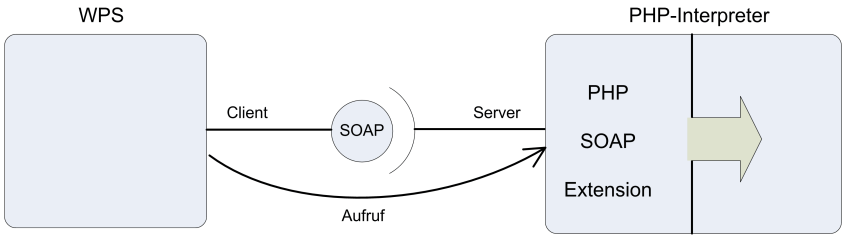

Abbildung 5.6: PHP-Web-Service zur Ausführung von PHP-Code

Die Funktionsweise ist dabei die Folgende: Dem PHP-Web-Service wird das auszuführende PHP-Snippet als Parameter übergeben. Der Web Service führt dieses Snippet über die spezielle, im PHP-Sprachumfang enthaltene Funktion `eval()` aus. Da PHP-Snippets ebenso wie Java-Snippets keine Rückgabewerte besitzen – einzig die Seiteneffekte sind interessant – ist die Implementierung des Web Services damit im Wesentlichen vollständig.

Zugriff auf Prozessvariablen über WSDL und SOAP
Um den Zugriff auf die Prozessvariablen bereitzustellen, muss ein spezieller Web Service erstellt werden, welcher Zugriff auf die Prozessinformationen hat. Dieser Web Service kann als Plugin für WPS realisiert werden, wie es in Abbildung 5.7 dargestellt ist. Theoretisch ist jedoch auch eine separate Komponente denkbar, sofern diese in der Lage ist, auf die Prozessinformationen zuzugreifen.

Auch in diesem Fall ist die Schnittstelle zwischen den beiden Komponenten via WSDL definiert und die Kommunikation wird über SOAP abgewickelt. Sowohl zum Laden als auch zum Speichern von Informationen bietet der Web Service Methoden an, auf welche bei der PHP-Ausführung zurückgegriffen wird.

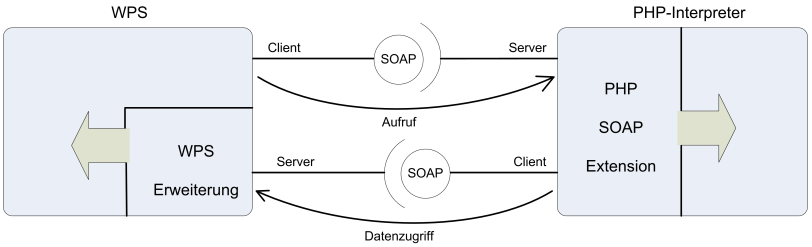

Abbildung 5.7: Zugriff auf Prozessvariablen mittels eines Web Service

Bewertung

Bezogen auf den in Abschnitt 5.2.1 vorgestellten Kriterienkatalog lassen sich für diese Lösung folgende Vor- und Nachteile konstatieren:

- **Kommunikation** In diesem Setup treten die beiden Parteien PHP-Interpreter sowie Workflow-Komponente als Web Services auf. Diese können per Definition über SOAP-Nachrichten miteinander kommunizieren. Allerdings ist diese Kommunikation verglichen mit den noch vorzustellenden Lösungen unverhältnismäßig teuer.
- **Ausnahme-Behandlung** Ausnahmen, die während der Ausführung des PHP-Codes auftreten, können nicht unmittelbar die geforderte Java RuntimeException verursachen. Eventuelle Ausnahmen können lediglich über spezielle Rückgaberwerte signalisiert werden.
- **Systemsicherheit** Die Sicherheit des Systems wird durch diesen Ansatz herabgesetzt: Für die Bereitstellung des PHP-Web-Service ist in jedem Fall die Einrichtung eines separaten Systems nötig[3], für welches zusätzliche Ports geöffnet werden müssen, was zusätzlichen Administrationsaufwand bedeutet und eine potentielle Schwachstelle des Systems darstellt.
- **Ausfallsicherheit** Auch die Ausfallsicherheit des Systems wird durch diesen Aufbau beeinträchtigt: Das Kernsystem wird von der korrekten Funktion des separaten Systems abhängig.
- **Performanz** Aus Performanz-Sicht ist diese Lösung ebenfalls suboptimal: Auf der Service-Seite kommt ein gewöhnlicher PHP-Interpreter zum Einsatz, welcher auf dem lokalen System eine ähnliche Performanz hätte. Zusätzlich zu einer lokalen Lösung muss jedoch der Aufruf des Interpreters sowie sämtliche Kommunikation über SOAP-Nachrichten abgewickelt werden, weshalb bei dieser Lösung eine deutlich schlechtere Performanz zu erwarten ist.

[3] Beispielsweise ein Apache Server

5.3.3 php/Java Bridge

Die folgenden Ausführungen beziehen sich auf die über `http://sourceforge.net/` verfügbare **php/Java Bridge**. Ein vergleichbares kommerzielles Produkt wird von der Firma Zend Technologies GmbH unter dem Namen **Java Bridge** vertrieben.

Kurz gesagt handelt es sich bei der php/Java Bridge um eine Technologie, die speziell zur Integration von PHP und Java entwickelt wurde. Laut der offiziellen Webseite[4] ist die php/Java Bridge im Wesentlichen ein „optimiertes, XML basiertes Netzwerkprotokoll, welches dazu verwendet werden kann, eine plattformabhängige Skripting-Engine (PHP) mit einer Java Virtual Machine zu verbinden“ [BK06].
Des Weiteren ist der Webseite zu entnehmen, dass mittels der php/Java Bridge schnell der Zugriff auf Java-Klassen aus PHP heraus realisiert werden kann. Der umgekehrte Weg – also der Zugriff aus PHP auf Java – soll genauso möglich sein.

Realisierung

Im Folgenden wird wiederum nur der konzeptuelle Aufbau betrachtet, insbesondere wird eine funktionierende Installation einer php/Java Bridge-Infrastruktur vorausgesetzt. Eine ausführliche Beschreibung, welche Schritte zur Installation der Bridge sowie zum Ausführen eines einfachen Beispiels nötig sind, können [Kan06] entnommen werden.
Nach der Installation der php/Java Bridge stehen sowohl in PHP als auch in Java spezielle API-Aufrufe zur Verfügung, mittels derer die Interaktion zwischen PHP und Java ermöglicht wird.

Integration
Zur Ausführung von PHP-Code wird eine entsprechende Methode über das Bridge-Protokoll aufgerufen. Der konzeptuelle Aufbau ist in Abbildung 5.8 dargestellt.

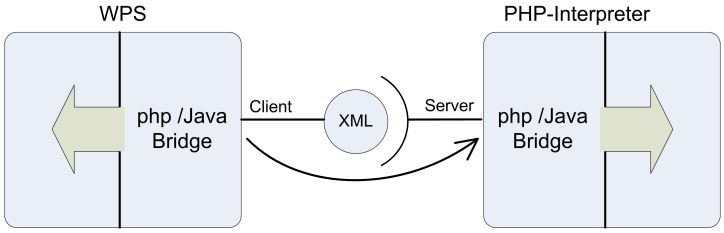

Abbildung 5.8: Auswertung von Snippet-Code über die php/Java Bridge

Die interne Umsetzung der PHP-Ausführung kann analog zu den Web Services geschehen: Über das von der Bridge verwendete XML-Protokoll wird eine spezielle PHP-Methode aufgerufen, welche als Argument den auszuführenden Snippet-Code erhält. Die Methode ruft die PHP-Methode `eval()` auf und übergibt dieser den Snippet-

[4] Zu finden unter `http://php-java-bridge.sourceforge.net/pjb/` (zuletzt besucht am 14.04.2007)

Code. Mittels der Methode `eval()` kann in PHP-Skripten beliebiger Skriptcode zur Auswertung an den PHP-Interpreter übergeben werden.

Zugriff auf Prozessvariablen
Der Zugriff auf die Prozessvariablen erfolgt über eine spezielle Java-Klasse, im Schaubild mit DAS (Data Access Service) bezeichnet. Dieser Aufbau ist in Abbildung 5.9 dargestellt.

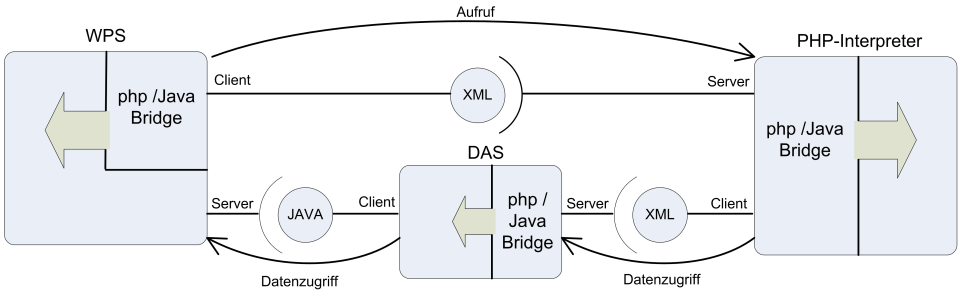

Abbildung 5.9: Zugriff auf Prozessinformationen über die php/Java Bridge

Der Ablauf ist in etwa der Folgende: Die PHP-Erweiterung greift über das Bridge-Protokoll auf die DAS-Klasse zu. Diese Klasse fragt über die normale Java-API die Informationen des Prozesskontextes ab. Aus technischen Gründen muss dabei der DAS auf einem separaten System, beispielsweise einem Apache Server, laufen. Dieser Umstand wird in dem Schaubild dadurch angedeutet, dass der DAS als separate Komponente eingezeichnet ist.

Bewertung

+ Kommunikation Die Kommunikation zwischen den beiden Parteien ist effizienter realisiert, als bei den Web Services: Es handelt sich zwar wiederum um ein XML-basiertes Datenaustausch-Format, jedoch ist dieses ausdrücklich speziell auf die Integration von PHP und Java zugeschnitten und entsprechend optimiert. Aus diesem Grund sind Geschwindigkeitsvorteile gegenüber der Web Service Lösung zu erwarten.

+ Ausnahme-Behandlung Die Behandlung von Ausnahmen wird direkt von der php/Java Bridge unterstützt. In PHP ab Version 5 können Java-Ausnahmezustände über das PHP-Konstrukt `catch (JavaException $ex)` abgefragt werden [Hab06].

- Systemsicherheit Zur Ausführung des PHP-Codes wird auch bei diesem Setup ein zweites System benötigt. Dieses System muss separat gewartet werden, um die Sicherheit des Systems zu gewährleisten.

- Ausfallsicherheit Das separate System erhöht die Wahrscheinlichkeit eines Ausfalls.

- Performanz Die Kommunikation wird über XML abgewickelt. Dieses Vorgehen gewährleistet die maximale Flexibilität der php/Java Bridge, bedeutet jedoch zeitaufwendiges Erzeugen und Einlesen von XML-Datensätzen.

5.3.4 Externer Prozess

Eine weitere Möglichkeit besteht darin, den PHP-Interpreter als externen Prozess zu starten. Theoretisch kann dafür die CGI-API[5] des PHP-Interpreters verwendet werden. Ab PHP in der Version 4.2 enthält der PHP-Interpreter jedoch eine Schnittstelle die besser für diesen Verwendungszweck geeignet ist: Die sogenannte **Command Line Interface (CLI)**-Schnittstelle[6]. Die CLI-Schnittstelle enthält eine ähnliche API wie die CGI-Schnittstelle, allerdings unterscheidet sie sich insbesondere durch die folgenden zwei Merkmale von der CGI-Schnittstelle:

- Die CLI-Schnittstelle erzeugt keine automatischen HTTP-Header für die Ausgabe.
- Fehlermeldungen werden als reiner Text ausgegeben, ohne HTML-Formatierungsanweisungen.

Realisierung

Integration
Der PHP-Interpreter wird unter Verwendung des Command Line Interface als externer Prozess gestartet. Direkt beim Start wird dem Interpreter dabei der PHP-Code übergeben, der ausgewertet werden soll. Dies kann über einen Kommanozeilen-Aufruf der folgenden Form bewerkstelligt werden:

```
php -r "echo 'hello world';"
```

Der Aufruf dieses Kommandos kann über das Java-Konstrukt `runtime.exec()` oder alternativ über die Klasse `ProcessBuilder`[7] vorgenommen werden. In beiden Fällen läuft der PHP-Interpreter anschließend in einem separaten Prozess des Betriebssystems.

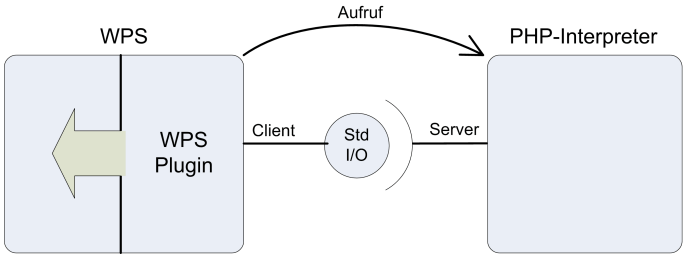

Abbildung 5.10: Zugriff auf den PHP-Interpreter über das Command Line Interface

Zugriff auf Prozessvariablen
Der Zugriff auf die Prozessvariablen kann über die Standard-Ein- und Ausgaben der beiden Prozesse erfolgen. Allerdings erfordert dies die Verwendung eines speziellen

[5] Die Abkürzung CGI steht für das **Common Gateway Interface**, welches speziell für die Entwicklung von Webapplikationen gedacht ist.
[6] Die offizielle Beschreibung der CLI-Schnittstelle kann unter `http://uk.php.net/features.commandline` abgerufen werden
[7] Diese beiden Ansätze werden auf der Webseite `http://java.sun.com/developer/JDCTechTips/2005/tt0727.html#2`(Zuletzt besucht 23.04.2007) miteinander verglichen.

Protokolls, welches den Zugriff auf die Standard-Ein- und Ausgaben sowie die Kommunikation zwischen den beiden Parteien regelt. Dazu ist eine Erweiterung für PHP nötig, welche die snippetseitig angeforderten Lade- und Speicheraktionen über die Standard-Ausgabe an die aufrufende Java-Klasse weiterreicht, sowie die Parameter entsprechend entgegennehmen kann.

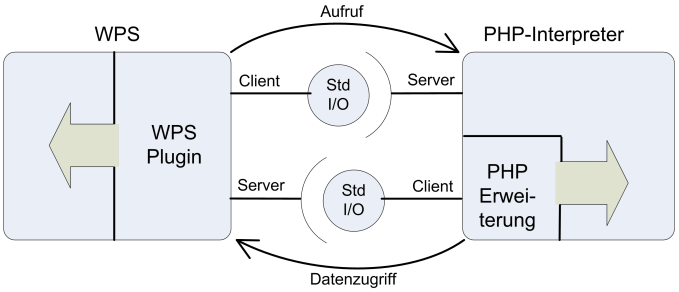

Abbildung 5.11: Zugriff auf Prozessinformationen über die Standard Ein-/Ausgabe

Bewertung

+ Kommunikation Die Kommunikation zwischen den beiden Komponenten erfordert die Entwicklung eines speziellen Kommunikationsprotokolls zwischen dem PHP-Interpreter auf der einen Seite und einer speziellen Java-Klasse auf der anderen Seite. Der Funktionsumfang des Protokolls ist relativ gering und kann mit Hilfe von XML schnell realisiert werden. Das Kommunikationsprotokoll ist hochspezialisiert und daher kann davon ausgegangen werden, dass diese Kommunikation noch effizienter und schneller abläuft als bei den bereits vorgestellten Lösungen.

- Ausnahme-Behandlung Die Ausnahme-Behandlung ist hier nicht in der gewünschten Weise möglich. Ausnahmezustände müssen über einen geeigneten Kommunikationsmechanismus über die Ein/-Ausgaben realisiert werden.

+ Systemsicherheit Die Systemsicherheit wird nicht beeinträchtigt.

+ Ausfallsicherheit Aufgrund der Tatsache, dass der abgespaltene Prozess auf dem selben System läuft entstehen keine zusätzlichen Abhängigkeiten. Die Ausfallsicherheit ist annähernd so hoch, wie die Ausfallsicherheit des Systems ohne die PHP-Erweiterung.

- Performanz Pro PHP-Snippet, das ausgeführt werden soll, muss bei dieser Lösung ein separater Prozess gestartet werden.

5.3.5 Java Native Interface

Die in diesem Vergleich technisch anspruchsvollste Lösung stellt die Ausführung des PHP-Interpreters über Java Native Interface - kurz JNI - dar.

Realisierung

PHP in der Version 5 enthält eine Bibliothek mit dem Namen `php5embed.lib`. Diese Bibliothek kann dazu verwendet werden PHP in andere Applikationen einzubinden. Konkret kann diese Bibliothek bei der Compilierung eines C/C++-Programms verlinkt werden und bietet so Zugriff auf eine große Anzahl PHP-spezifischer API-Aufrufe. Einige dieser API-Aufrufe beschäftigen sich mit der Auswertung von PHP-Code.

Integration von PHP über Java Native Interface
Da für Java-Programme keine Möglichkeit besteht, diese Programmbibliothek zu verwenden, muss ein Umweg zur Integration von PHP gewählt werden: Die Programmbibliothek wird in eine Dynamic Link Library[8] – kurz DLL – mit eincompiliert. Die DLL bietet via JNI Methoden nach außen an, die alle relevanten PHP-API-Aufrufe in Java zur Verfügung stellen. Diese DLL wird dann per JNI in die Java Laufzeitumgebung (JRE) geladen. Dieser Aufbau ist graphisch dargestellt in Abbildung 5.12.

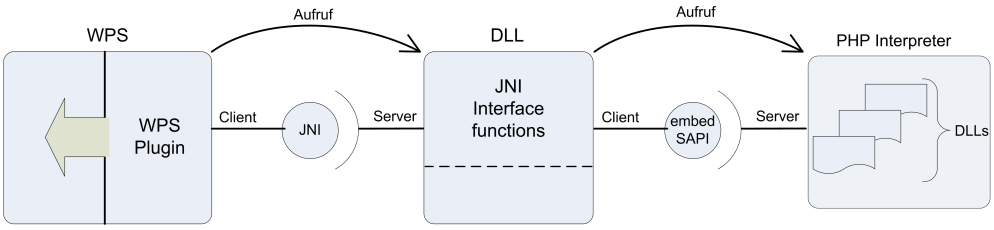

Abbildung 5.12: Integration von PHP per Java Native Interface

Auf die angesprochene Weise wird die Java-Laufzeitumgebung in die Lage versetzt alle für die PHP-Ausführung nötigen PHP-API-Aufrufe durchzuführen, also insbesondere den PHP-Interpreter dazu zu veranlassen, PHP-Code auszuwerten. Streng genommen wird dabei nicht auf den PHP-Interpreter, sondern eine Reihe von DLLs zurückgegriffen, die in ihrer Gesamtheit den Funktionsumfang von PHP ausmachen. Dieser Umstand wird in dem Schaubild durch das Einzeichnen der DLLs in den PHP-Interpreter angedeutet.

Zugriff auf Prozessvariablen über PHP-Erweiterung
Zum Abfragen und Speichern von Informationen sind Callback-Funktionen aus dem PHP-Skript zurück in den Java-Kontext nötig. Die einzige Möglichkeit, diese bereit zu stellen, liegt darin, für die PHP-Ausführungskomponente eine Erweiterung vorzusehen, die diese Callback-Funktionen in PHP verfügbar macht.
PHP bietet die Möglichkeit, Erweiterungen während der Startphase des PHP-Interpreters oder auch während der Laufzeit eines Skriptes zu laden. Um eine Erweiterung während der Startphase des Interpreters zu laden muss ein entsprechender Eintrag in der Konfigurationsdatei `php.ini` vorgenommen werden.
Erweiterungen für PHP können unter Windows ebenfalls in Form einer DLL bereitgestellt werden. Es bietet sich daher an, die Erweiterung für PHP in die bereits existierende

[8] Die Entwicklung sämtlicher in dieser Arbeit vorgestellten Komponenten erfolgt unter dem Betriebssystem Microsoft Windows XP. Unter UNIX-artigen Betriebssystemen werden diese Komponenten als Shared Objects (– kurz SO –) bezeichnet.

DLL, die zum Zugriff auf die PHP-API erstellt wurde, zu integrieren. Damit ergibt sich für die Kommunikation von PHP zu Java der in Abbildung 5.13 dargestellte Aufbau:

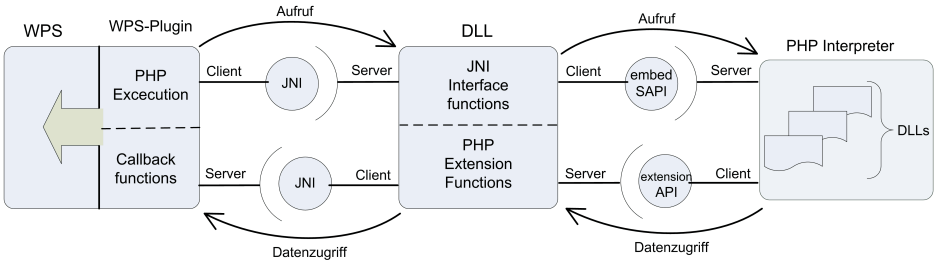

Abbildung 5.13: Kommunikation per PHP-Erweiterung

Der PHP-Interpreter lädt während der Startup-Phase eine DLL. Diese DLL stellt über eine spezielle PHP-spezifische Funktionstabelle der PHP-Laufzeitumgebung Funktionen zur Verfügung, die in den Skripten verwendet werden können. Diese Funktionen bieten den Zugriff auf den in der Java-Laufzeitumgebung gespeicherten Prozesskontext auf die folgende Weise: Die Funktionen der PHP-Erweiterung rufen über Java Native Interface spezielle Java-Callback-Funktionen auf, die von der PHP-Erweiterung des WPS zur Verfügung gestellt werden. Diese greifen auf die normale – auch in Java-Snippets verwendete – Weise auf die gewünschten Informationen des Prozesskontextes zu. Das Ergebnis des Aufrufs wird über die die DLL zurück gereicht an den PHP-Interpreter. Mittels dieser Callback-Funktionen kann das PHP-Snippet zu beliebigen Zeitpunkten lesend und schreibend auf den Prozesskontext und die Prozessvariablen zugreifen.

Bewertung

+ Kommunikation Die Kommunikation kann sehr effizient vonstatten gehen: Die beiden Bestandteile PHP und Java kommunizieren miteinander über Methodenaufrufe. Für die Parameterübergabe ist in diesem Fall weder der Umweg über ein Netzwerk noch über spezielle Betriebssystemressourcen (wie beispielsweise Standard Ein-/Ausgabe) nötig. Natürlich ist auch bei dieser Lösung eine Datentypkonvertierung nötig, welche jedoch vergleichsweise effizient durchgeführt werden kann.

+ Ausnahme-Behandlung Die Interaktion über Java Native Interface birgt einen weiteren entscheidenden Vorteil: Über Java Native Interface können im Fehlerfall direkt Java RuntimeExceptions erzeugt werden: JNI bietet Zugriff auf die Klasse `RuntimeException` sowie einen speziellen API-Aufruf, mit dem eine solche erzeugt werden kann. Ein spezielles 'Protokoll' über den Rückgabewert kann somit entfallen.

+ Systemsicherheit Die Systemsicherheit wird durch diese Vorgehensweise nicht beeinträchtigt. Es wird weder über ein Netzwerkprotokoll kommuniziert, noch ein externer Prozess gestartet. Das System bietet nach der Integration von PHP keine wesentlich vergrößerte Angriffsfläche.

- Ausfallsicherheit Im Gegensatz zu allen vorgestellten Lösungen wird bei dieser Lösung kein zusätzlicher Prozess benötigt. Die Interpretation von PHP läuft im gleichen Prozess ab. Das Problem an dieser sehr engen Integration ist die Tatsache, dass Fehler in der PHP-Ausführungskomponente bei dieser Lösung zum völligen Absturz des gesamten Systems führen können. Bei allen anderen vorgestellten Lösungen ist dies nicht der Fall: Im Fehlerfall fällt lediglich die PHP-Ausführung aus, das System als solches bleibt ansprechbar.

+ Performanz Es wird keine Kommunikation über das Netzwerk benötigt und kein separater Prozess benötigt. Es ist zu erwarten, dass dies die performanteste der vorgestellten Lösungen darstellt.

5.4 Zusammenfassung und Entscheidung

Zum Abschluss dieses Kapitels werden die vorgestellen Möglichkeiten zur Integration des PHP-Interpreters hier abschließend vergleichend bewertet und die Entscheidung für eines der vorgestellten Modelle getroffen.

Aus der Übersicht in Tabelle 5.1 wird deutlich, dass die auf JNI basierende Lösung die meisten Vorteile bietet. Etwas differenzierter lässt sich folgendes Fazit über die Tauglichkeit der einzelnen Realisations-Möglichkeiten feststellen:

	Service	P/J Bridge	ext. Prozess	JNI
Kommunikation	-	+	-	+
Ausnahme-Behandlung	-	+	-	+
Systemsicherheit	-	-	+	+
Ausfallsicherheit	-	-	+	-
Performanz	-	-	-	+

Tabelle 5.1: Vergleich der verschiedenen Umsetzungsmöglichkeiten

Web Service
Die Realisierung über einen Web Service ist mit vergleichsweise geringen Mitteln möglich:

- Es wird keine Erweiterung für BPEL benötigt.
- PHP-seitig kann die existierende Erweiterung für Web Services ohne Modifikationen verwendet werden.
- Die Erweiterung des WPS beschränkt sich auf die Bereitstellung eines Web Service mit Zugriff auf den Prozesskontext.

Abgesehen von diesem Vorteil scheitert diese Lösung an Performanz sowie Sicherheitsaspekten: Sowohl der Aufruf des PHP-Interpreters als auch jeglicher Variablenzugriff setzt die Kommunikation über den Web-Service-Stack voraus. Außerdem muss ein weiteres System – beispielsweise ein Apache Server – gewartet werden.

Mit den oben angesprochenen Eigenschaften eignet sich die Realisiation über Web Services hervorragend für die Erstellung eines Prototypen. Für den Produktiveinsatz stellt sie jedoch keine geeignete Lösung dar.

php/Java Bridge
Eine deutliche Verbesserung gegenüber den Web Services stellt die Verwendung der php/Java Bridge dar. Zum einen wird die Kommunikation über ein optimiertes XML-Format vorgenommen, woraus Geschwindigkeitsvorteile zu erwarten sind. Des Weiteren ist hier ein Mechanismus zur Ausnahmebehandlung integriert, der über die Grenzen von PHP oder Java hinaus funktioniert. Auf der anderen Seite wird für diese Lösung ein separater Server benötigt, was die Ausfallsicherheit sowie die Systemsicherheit herabsetzt. Abgesehen davon wird die Lösung dadurch zusätzlich abhängig von externen Entwicklern für das verwendete Integrationsframework (php/Java Bridge). und es treten zusätzlich zu klärende Lizenzfragen auf. Diese Umstände lassen auch diese Lösung als suboptimal erscheinen.

Externer Prozess
Eine passendere Lösung für das Problem stellt die Lösung dar, in dem der PHP-Interpreter als externer Prozess über das Command Line Interface des Interpreters eingebunden wird. Der Vorteil gegenüber der Bridge-Lösung besteht darin, dass die JVM, welche den PHP-Interpreter aufgerufen hat, direkt über den Ein/-Ausgabestrom mit dem Interpreter verbunden ist. Außerdem entstehen abgesehen vom PHP-Interpreter keine zusätzlichen Abhängigkeiten. Dennoch ist für die Ausführung von PHP-Code für jedes Snippet das Starten eines separaten PHP-Interpreters nötig, was sich negativ auf die Performanz auswirkt. Zudem kann die Ausnahmebehandlung lediglich über eine spezielle Kommunikation über den Ein/Ausgabestrom realisiert werden, was ebenfalls negativ zu bewerten ist.

Java Native Interface
Gegen die Lösung mittels des JNI spricht die Tatsache, dass dafür der größte Entwicklungsaufwand zu erwarten ist. Abgesehen davon birgt die Verwendung des JNI als einzige Lösung die Gefahr bei Programmierfehlern den WPS vollständig zum Absturz zu bringen.
Auf der anderen Seite vereint die JNI-basierte Lösungen einige Vorteile:

- Performanz: Es ist zu erwarten, dass diese Lösung die performanteste Lösung darstellt, da für die Ausführung von PHP-Code kein separater Prozess benötigt wird.
- Abhängigkeit: Die JNI-basierte Lösung reduziert die Abhängigkeit von zusätzlich benötigten externen Produkten auf ein Minimum. Es wird lediglich das JNI sowie die PHP embed SAPI benötigt.

FAZIT
Aufgrund der oben aufgeführten Punkte stellt die Lösung unter Verwendung des **Java Native Interface** die geeignetste Lösung für die dauerhafte Realisierung der PHP-Snippets dar. Die Integration des PHP-Interpreters über das JNI wird im folgenden Kapitel ausführlicher beschrieben.

6

Konzept und Implementierung

In den vorausgegangenen Ausführungen wurde der konzeptuelle Aufbau der WPS-Erweiterung bereits grundlegend vorgegeben. Der PHP-Interpreter wird über Java Native Interface mit der Java-Laufzeitumgebung verbunden. Wie im vorausgegangenen Kapitel dargestellt wurde, sind dazu zwei voneinander unabhängige Teilprobleme zu lösen:

Integration Dieser Punkt bezieht sich auf zwei Sachverhalte:

1. **BPEL-Erweiterung**: Die PHP-Snippets müssen in BPEL-Dokumente eingebettet werden können. Dies kann analog zu den Java-Snippets geschehen und ist in Abschnitt 6.1 beschrieben.
2. **WPS-Erweiterung:** Der in den PHP-Snippets enthaltene PHP-Code muss von WPS ausgewertet werden können. Dazu muss der PHP-Interpreter über ein WPS-Plugin in die Prozess-Laufzeitumgebung integriert werden. Die Entwicklung dieses Plugins wird in den Abschnitten 6.2 und 6.3 beschrieben.

Kommunikation Das PHP-Snippet muss in der Lage sein, zu jedem beliebigen Zeitpunkt auf die in dem Geschäftsprozess definierten Variablen sowie den Kontext des Geschäftsprozesses zugreifen zu können. Zu diesem Zweck muss eine Erweiterung für die PHP-Laufzeitkomponente bereitgestellt werden. Diese Erweiterung wird in Abschnitt 6.4 beschrieben.

Wie diese beiden Erweiterungen konzeptuell mit ihrem jeweiligen Partner interagieren, wurde bereits in Abschnitt 5.3.5 dargestellt. In diesem Kapitel wird die Interaktion zwischen diesen Partner konkretisiert sowie einzelne Details der konkreten Implementierung vorgestellt.

6.1 BPEL-Erweiterung

Der BPEL-Standard sieht die Integration von Programmiersprachen in BPEL nicht vor. Daher ist für deren Integration eine Erweiterung zu BPEL nötig. Für die Programmiersprache Java gibt es die bereits in Abschnitt 3.6 vorgestellte, IBM-proprietäre Erweiterung in Form der Java-Snippets. Die Integration von PHP in BPEL-Dokumenten kann sich an der Integration von Java-Snippets in BPEL-Dokumenten orientieren. Wie in Abschnitt 3.6 gezeigt wurde, sind die Java-Snippets eine Spezialform des `<invoke>`. Aus diesem Grund werden auch PHP-Snippets als Spezialform des `<invoke>` realisiert. Natürlich kann PHP nicht auf exakt die gleiche Weise wie Java in BPEL integriert werden, es muss in der Prozessdefinition klar zwischen PHP- und Java-Snippets unterschieden werden. Daher sind die folgenden Schritte nötig:

- Ein neues Element muss bereitgestellt werden, welches den PHP-Code transportiert. Der Name dieses Elementes kann willkürlich gewählt werden und wurde auf `BpelPHPActivity` festgelegt.
- Damit WPS beim Installieren der Prozessdefinition dieses neue Element erkennt und korrekt überprüfen und behandeln kann, muss dieses Element eine Namespace-Deklaration tragen. Der Namespace der dieses Element deklariert kann ebenfalls frei gewählt werden und wurde auf `http://bpe.ibm.com/customactivities/php` festgelegt.

Der BPEL-Code zur Integration von PHP sieht dementsprechend folgendermaßen aus:

Listing 6.1: Integration von PHP-Snippets in BPEL

```
1 <invoke ...>
2  <bpelphp:BpelPHPActivity
3    xmlns:bpelphp="http://bpe.ibm.com/customactivities/php">
4    <!-- hier wird der PHP-Code eingebettet -->
5    <![CDATA[ echo 'hello world'; ]]>
6   </bpelphp:BpelPHPActivity>
7 </invoke>
```

Dazu sind die folgenden Anmerkungen zu machen:

- Zeile 1: Wie schon bei den Java-Snippets, so können auch bei PHP-Snippets sämtliche Pflichtattribute des `<invoke>` leer bleiben, da es sich nicht um einen Serviceaufruf handelt. Der Übersicht halber wurden diese Attribute in Listing 6.1 nicht abgedruckt.
- Zeile 2: Das `<invoke>`-Element trägt als direktes Kind-Element ein Element mit dem Namen `BpelPHPActivity`. Dieses Element enthält den Code des PHP-Snippets, eingepackt in ein CDATA-Element (Zeile 5).
- Zeile 3: Das neue Element `BpelPHPActivity` trägt den Namespace `http://bpe.ibm.com/customactivities/php`, der zur Validierung sowie korrekten Installation der Prozessdefinition benötigt wird.

Die letzten beiden Punkte werden im Folgenden kurz erläutert:

CDATA-Element

Beim Parsen von XML-Code gibt es einige Zeichen, die spezielle Bedeutung für den Parser haben: Diese sind insbesondere die beiden Zeichen > sowie <, aber auch einige weitere Zeichen wie beispielsweise " oder ’. Grundsätzlich wird beim Parsen sämtlicher Text der XML-Datei berücksichtigt, was beim Einbetten von Programmcode in den XML-Code eine erhebliche Fehlerquelle darstellen kann, wie aus Listing 6.2 ersichtlich ist:

Listing 6.2: Fehlerbehafter XML-Code

```
1 ...
2  <bpelphp:BpelPHPActivity ...>
3    if (\$salary < 1000) {...}
4   </bpelphp:BpelPHPActivity>
5 ...
```

In Zeile 3 wird der PHP-Code ohne besondere Maßnahmen in den XML-Code eingebettet. Durch das Sonderzeichen < in der PHP-Vergleichsoperation würde jedoch bei der Arbeit mit dem XML-Dokument ein Fehler entstehen, da der XML-Parser das Zeichen < als Begin eines neuen Tags auffasst, was es an dieser Stelle jedoch nicht darstellt. Dieser Fehler kann dadurch vermieden werden, dass der PHP-Code in einen Teil des XML-Dokumentes verpackt wird, den der XML-Parser ignoriert. Speziell zu diesem Zweck wurde das CDATA[1]-Element entworfen. Sämtliche Zeichen zwischen dem öffnenden Token `<![CDATA[` und dem schließenden Token `]]>` werden vom XML-Parser vollständig ignoriert. Somit werden unnötige Fehler beim Parsen des XML-Dokumentes vermieden.

Namespace-Deklaration

Für XML-Dokumente gibt es zwei unterschiedlich aussagekräftige Gütekriterien: XML-Dokumente sind **wohlgeformt** (oder auch **well-formed**), wenn sie sich an einige wenige Konventionen halten, beispielsweise die Tatsache, dass Elemente grundsätzlich ein schließendes Tag haben und dass sie korrekt ineinander geschachtelt sind. XML-Dokumente sind darüber hinaus **gültig** (oder auch **valid**), wenn sie wohlgeformt sind und sich zudem an die Regeln halten, die in einer XML-Schema-Definition[2] (XSD) definiert sind[3]. Wie in Abschnitt 3.4 bereits erwähnt wurde, wird für jeden Prozess, der

[1] CDATA ist die Kurzbezeichnung für „Character Data“

[2] Eine XML-Schema-Datei definiert den formalen Aufbau einer bestimmten Art von XML-Datei. Alternativ kann auch eine so genannte ’Document Type Definition’ (DTD) verwendet werden. Mehr Informationen über XSDs und XML bietet beispielsweise **’The basics of using XML Schema to define elements’** [RD00].

[3] Mehr Informationen zum Thema wohlgeformte und gültige XML-Dokumente sind beispielsweise unter `http://www.w3schools.com/xml/xml_dtd.asp` zu erfahren oder können dem offiziellen Standard ’*Extensible Markup Language (XML) 1.0*’ [BPSM+06] entnommen werden.

auf WebSphere Process Server installiert wird, eine Prüfung auf Gültigkeit des BPEL-Dokumentes – auch Validierung genannt – durchgeführt. Damit diese Validierung fehlerfrei funktioniert muss WPS eine entsprechende XML-Schema-Definition für jeden Teil des Dokumentes auffinden können. Der Großteil des Dokumentes wird dabei mit Hilfe der XSD des BPEL-Standards, auf die im Kopf des Dokumentes verwiesen wird, validiert. Die oben gezeigte Erweiterung kann nur unter zwei Bedingungen korrekt validiert werden:

- Die Erweiterung muss eine entsprechende Namespace-Deklaration tragen.
- Die Definition der gemäß diesem Namespace gültigen Dokumente – sprich die Inhalte einer entsprechenden XML-Schema-Datei – muss WebSphere Process Server bekannt sein.

Wie im Abschnitt 6.2 gezeigt wird, ist es Aufgabe des noch vorzustellenden WPS-Plugins, diese XSD bereitzustellen. Die für die oben gezeigte Erweiterung nötige XSD hat den in Listing 6.3 aufgeführten Aufbau.

Listing 6.3: XSD der BPEL-Erweiterung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema
3    xmlns:bpelphp="http://bpe.ibm.com/customactivities/php"
4    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5    targetNamespace="http://bpe.ibm.com/customactivities/php">
6   <xsd:element name="BpelPHPActivity" type="xsd:string"/>
7 </xsd:schema>
```

Die wichtigen Informationen, die diese Schemadatei trägt sind:

- Die Angabe **targetNamespace** in Zeile 5 gibt an, welcher Namespace durch diese Schemadatei definiert wird. In diesem Fall ist dies der, durch die Erweiterung verwendete, Namespace `http://bpe.ibm.com/customactivities/php`.
- Die Angabe `<xsd:element ...>` in Zeile 6 der XSD-Datei definiert das einzige innerhalb dieser Erweiterung gültige Element: Dieses Element trägt den Namen `BpelPHPActivity` und enthält Daten vom Typ String – nämlich den PHP-Code.

6.2 Erweiterung des WebSphere Process Servers

Die im vorausgegangenen Abschnitt beschriebene Erweiterung von BPEL lehnte sich stark an die für die Java-Snippets vorgesehene BPEL-Erweiterung an. Für die Umsetzung von PHP-Snippets in WPS ist solch eine starke Anlehnung aus den folgenden Gründen nicht möglich:

1. Die Java-Snippets sind fester Bestandteil des WPS, PHP-Snippets dagegen sind bis auf weiteres lediglich ein Prototyp, der für Testzwecke verwendet wird.
2. Die Java-Snippets verwenden die gleiche Programmiersprache – nämlich Java –

wie die verwendeten BPEL-Werkzeuge. Eine Integration von Java-Snippets ist somit auf vergleichsweise einfachem Weg möglich.

Aus den oben genannten Gründen muss ein Plugin für WPS entwickelt werden, welches WPS in die Lage versetzt, PHP-Code auszuführen. Die Ausführungen in diesem Kapitel sind der Übersicht halber in zwei separate Aspekte untergliedert:

- Der Abschnitt **6.2.1 Realisierung eines Plugins für WPS** erläutert, auf welche Weise WPS dahingehend erweitert werden kann, dass neue BPEL-Elemente korrekt erkannt und behandelt werden können.
- In dem Abschnitt **6.2.2 Details des WPS-Plugins** wird detaillierter auf einzelne Aspekte des Plugins eingegangen.

6.2.1 Realisierung eines Plugins für WPS

Wie zuvor angesprochen muss die BPEL-Erweiterung eine Namespace-Deklaration tragen, mit deren Hilfe WPS bei der Installation eines Geschäftsprozesses die nötigen Schritte Validierung sowie Installation vornehmen kann. Für die Integration des Plugins in WPS sind die folgenden zwei Punkte interessant:

Plugin-Schnittstelle Wie muss das Plugin für WPS aufgebaut sein, damit WPS erfolgreich damit arbeiten kann? Etwas konkreter ausgedrückt: Welche Funktionen muss das Plugin WPS zur Verfügung stellen?

Auffindung Wie wird das Plugin, das für die BPEL-Erweiterung zuständig ist, aufgefunden?

Diese beiden Fragen werden durch die folgenden beiden Abschnitte beantwortet:

Plugin-Schnittstelle

Bei den PHP-Snippets handelt es sich um eine neue Basisaktivität (siehe 3.2.2) für BPEL-Prozesse. Um diese neue Basisaktivität WPS bekannt zu machen, kann WPS über eine spezielle interne Schnittstelle erweitert werden. Diese Schnittstelle besteht aus zwei Teilen, welche die Namen **CustomActivityPlugin** beziehungsweise **CustomActivityValidationPlugin** tragen. Die Zuständigkeiten der beiden Teile sind die Folgenden:

- Das **CustomActivityValidationPlugin** wird während der Validierungsphase des Prozessmodells aufgerufen. Es ist dafür zuständig die entsprechende XSD für den neu eingeführten Prozessabschnitt bereitzustellen. Außerdem kann durch das `CustomActivityValidationPlugin` eine weitreichende Überprüfung des Prozessmodells mittels des Eclipse Modelling Frameworks (EMF) durchgeführt werden.
- Das **CustomActivityPlugin** wird sowohl bei der Installation des Prozesses als auch bei der Ausführung des Prozesses benötigt. Während der Installation des Prozesses ist es dafür zuständig, alle Informationen, die für die spätere Ausführung der Aktivität benötigt werden, aus dem BPEL-Code zu extrahieren.

Zur Ausführungszeit wird dieses Plugin erneut herangezogen, um die Aktion, die dieser Aktivität entspricht, auszuführen.

Durch die Implementierung dieser beiden Schnittstellen wird ein gültiges Plugin für WPS erzeugt, welches für die Installation und Ausführung einer neuen Aktivität zuständig ist. Um dieses Plugin auf WPS zu installieren reicht es aus, es in ein Java-Archive (JAR) zu verpacken und in einem der Verzeichnisse, in denen WPS nach Laufzeitkomponenten sucht, abzulegen. Verzeichnisse, die dafür in Frage kommen, sind die Folgenden:

- Das **lib**-Verzeichnis: In diesem Verzeichnis liegt eine Vielzahl von Java Archiv-Dateien, welche den wesentlichen Funktionsumfang von WPS bereitstellt.
- Das **classes**-Verzeichnis: Dieses Verzeichnis ist für gewöhnlich leer und wird lediglich für temporäre Tests verwendet. Dateien in diesem Verzeichnis überschreiben eventuell im lib-Verzeichnis enthaltene Dateien. Dies wird über die Suchreihenfolge des WPS erreicht.

Da es sich bei der entwickelten Erweiterung um eine experimentelle Komponente handelt, ist das classes-Verzeichnis somit der richtige Speicherort für das Java Archiv.

Auffinden des Plugins

Durch Ablegen des noch zu entwickelnden JARs in einem dieser Verzeichnisse wird das Plugin in WPS verfügbar und kann im Bedarfsfall ausgeführt werden. Allerdings ist bisher unklar, auf welche Weise WPS herausfindet, welches Plugin für die unbekannte BPEL-Passage zuständig ist. Um diese Zuordnung vornehmen zu können, wird ein Namespaces-Mapping durchgeführt. Dabei wird folgendermaßen vorgegangen:
Die im BPEL-Dokument angetroffene Namespace-URL der Aktivität wird mit Hilfe der in Abbildung 6.1 dargestellten Regeln in einen Java-Package-Name umgewandelt[4].

Die einzelnen Regeln des Namespace-Mappings lauten:

1. Die Schema-Angabe der Namespace-URL der BPEL-Erweiterung wird verworfen.
2. Die Authority-Komponente wird anhand des Abtrenners Punkt (.) aufgespalten und in der umgekehrten Reihenfolge notiert. Dies ergibt den ersten Teil des Java-Package-Names.
3. An den ersten Teil des Package-Names wird die Pfad-Angabe der Namespace-URL angehängt, wobei die Slashes (/) durch Punkte (.) ersetzt werden.

Vorrausgesetzt, der für die BPEL-Erweiterung verwendete Namespace ist eindeutig, so ergibt sich auf diese Weise ein ebenfalls eindeutiger Package-Name für das Java-Package. Für den konkreten Fall der oben erwähnten Namespace-URL `http://bpe.ibm.com/customactivities/php` lautet der entsprechende Java-Package-Name nach Anwendung der beschriebenen Regeln `com.ibm.bpe.customactivities.php`.
Durch das vorgestellte Mapping kann WPS in dem Moment, in dem eine unbekannte

[4]Die Bezeichnungen für die einzelnen Bestandteile einer URL sind dem RFC 3986 Uniform Resource Locators –zu finden auf `http://tools.ietf.org/html/rfc3986#section-3` (zuletzt besucht am 27.04.2007) – entnommen

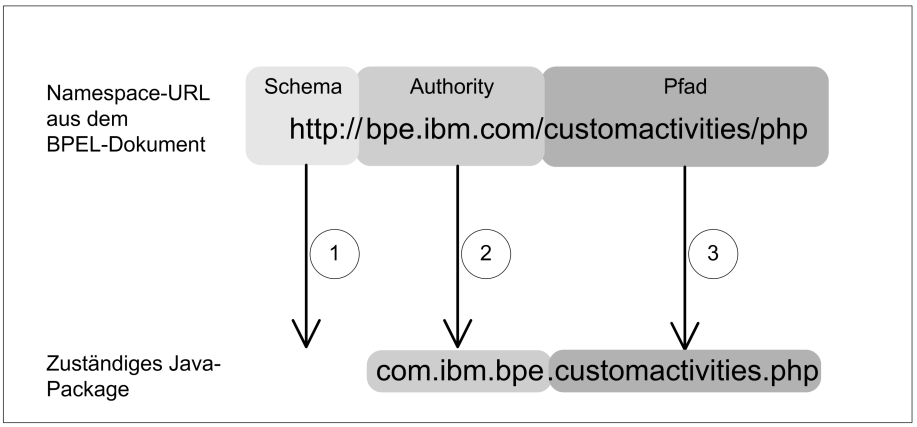

Abbildung 6.1: Namespace-Mapping zum Auffinden des zuständigen Plugins

Erweiterung angetroffen wird, anhand der angegebenen Namespace-URL der Erweiterung, das für diese Erweiterung zuständige Java-Package ermitteln.

6.2.2 Details des WPS-Plugins

Wie im Abschnitt 6.2.1 bereits dargstellt wurde, kann durch die Implementation der Schnittstellen `CustomActivityValidationPlugin` sowie `CustomActivityPlugin` eine Erweiterung für WPS realisiert werden. Diese beiden Schnittstellen wurden in den Klassen `BPELPHPActivityValidator` respektive `BPELPHPActivity` implementiert.
Der Funktionsumfang, der durch diese Schnittstellen vorgeschrieben wird, wird im Folgenden vorgestellt. Dabei wird bewusst darauf verzichtet die Parameter der einzelnen Methoden anzugeben, um eine möglichst übersichtliche Darstellung des Funktionsumfangs zu ermöglichen. Die Details der Implementierung können dem Quellcode, der der Arbeit beigelegt ist, entnommen werden.

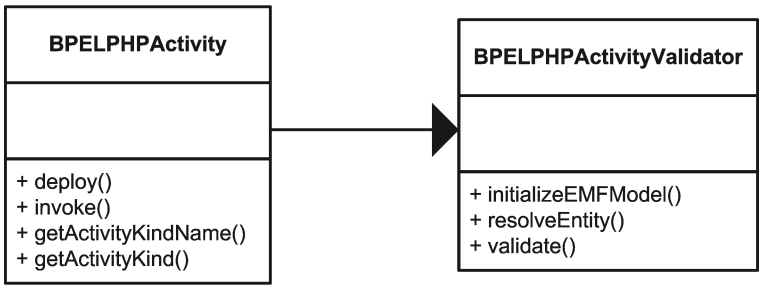

Abbildung 6.2: Aufbau der WPS-Pluginschnittstelle

Wie aus Abbildung 6.2 ersichtlich ist, bietet die `CustomActivityValidationPlugin`-Schnittstelle dem WPS die folgenden Methoden an, deren Zuständigkeit wie folgt ist:

initializeEMFModel() baut ein EMF-Modell auf, welches beispielsweise für weitreichende Gültigkeitsprüfungen des Prozess-Modells verwendet werden kann.

resolveEntity() ist dafür zuständig, die XML-Schema-Definition zur Verfügung zu stellen, mit dessen Hilfe der Teil der BPEL-Prozessdefinition, der mit der Erweiterung zusammenhängt, syntaktisch validiert werden. Diese Validierung wurde bereits in Abschnitt 6.1 angesprochen.

validate() führt eine semantische Überprüfung der Prozessdefinition durch. Dazu wird eine EMF-Repräsentation der Prozessdefinition verwendet.

Ebenfalls aus Abbildung 6.2 ist ersichtlich, dass das `CustomActivityPlugin` von `CustomActivityValidationPlugin` erbt und zusätzlich die folgenden Methoden anbietet:

deploy() bekommt als Eingabe alles, was zwischen dem startenden und dem endenden Tag der zugehörigen Aktivität liegt. Für den Fall der PHP-Snippets also alles, was zwischen `<invoke>` und `</invoke>` liegt.
Als Rückgabe liefert die Funktion alle Teile, die für die Ausführung der Aktivität zur Laufzeit des Prozesses nötig sind. Im Falle der PHP-Snippets handelt es sich hierbei um den enthaltenen PHP-Code.

invoke() wird zur Laufzeit des Prozesses aufgerufen. Diese Methode bekommt als Eingabe die durch die `deploy()`-Methode festgelegte Information. Zusätzlich hat sie beliebigen Zugriff auf die zur Laufzeit des Prozesses vorhandene Information.

getActivityKindName() gibt den Namen der CustomActivity zurück

getActivityKind() gibt den Typ der CustomActivity zurück.

Von spezieller Bedeutung für die folgenden Ausführungen ist die Methode `invoke()`, welche zur Laufzeit des Prozesses dafür sorgt, dass der übergebene PHP-Code ausgeführt wird. Der wesentliche Teil der Implementierung dieser Methode ist in Listing 6.4 abgedruckt.

Listing 6.4: Implementation der Methode invoke() (Auszug aus BPELPHPActivity.java)

```
1  public class BpelPHPActivity extends BpelPHPActivityValidator implements
2     CustomActivityPlugin {
3
4    ...
5
6    public void invoke(Serializable phpCode, ActivityContext activityCtx)
7        throws ApplicationFaultException, StandardFaultException,
8        RuntimeFaultException {
9        ...
10     PHPIntegration integrator = new PHPIntegration(activityCtx);
11     integrator.executePHPSnippet((String) phpCode, ...);
12   }
13 }
```

Wie aus dem Listing zu erkennen ist, bedient sich die Methode `invoke()` für die Ausführung des PHP-Codes einer weiteren, speziell für diesen Zweck entwickelten Klasse mit Namen `PHPIntegration`. Diese Abhängigkeit wird in Abbildung 6.3 angedeutet.

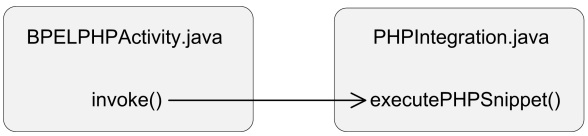

Abbildung 6.3: Abhängigkeit der Methode invoke()

Die Klasse `PHPIntegration`, sowie alle weiteren Maßnahmen, die zur Integration des PHP-Interpreters nötig sind, werden im Folgenden vorgestellt.

6.3 Integration des PHP-Interpreters über JNI

Vorbemerkung Der Bestandteil *native* in der Bezeichnung *Java Native Interface* bezieht sich auf die Tatsache, dass die aufgerufenen Methoden nicht, wie bei Java üblich, in der Java Virtual Machine ablaufen, sondern direkt auf dem Prozessor. In Ermangelung eines passenden deutschen Wortes wird im Folgenden an den entsprechenden Stellen von 'nativen' Methodenaufrufen gesprochen.

Bis zu diesem Punkt der Beschreibung wurden Lösungen für folgende Teilprobleme vorgestellt: Für BPEL wurde eine Erweiterung definiert, welche den PHP-Snippet-Code transportieren kann. Des Weiteren wurde ein Plugin für WPS vorgestellt, das mit der BPEL-Erweiterung umgehen kann, diese also korrekt erkennt und während der Installation des Prozesses die nötigen Schritte durchführt.
Damit zur Laufzeit des Prozesses der PHP-Code ausgewertet werden kann, muss in diese Erweiterung der PHP-Interpreter integriert werden. Dies geschieht mit Hilfe der speziell zu diesem Zweck entwickelten Java-Klasse `PHPIntegration`.
In Abschnitt 5.4 wurde festgestellt, dass das Java Native Interface die beste Lösung zur Integration des PHP-Interpreters darstellt. Die Klasse `PHPIntegration` verwendet daher das JNI für den Aufruf des PHP-Interpreters. Dazu sind die folgenden Punkte anzumerken:

- Das JNI lädt die nativen Funktionen zur Laufzeit des Programms, aus einer dynamischen Bibliothek (DLL unter Windows). Das bedeutet, dass zur Laufzeit des Programmes eine .dll-Datei bereitgestellt werden muss, in welcher die Implementationen der nativen Funktionen aufgefunden werden können. Die Details zur Verwendung des JNI werden in Abschnitt 6.3.1 vorgestellt.
- Die PHP-Distribution bietet eine spezielle **embed SAPI**[5], welche die Integration des PHP-Interpreters in C-Anwendungen ermöglicht. Diese API bietet in C-Programmen eine Vielzahl von Funktionen. Unter anderem kann mittels dieser API über den Aufruf der Funktion `zend_eval_string()` die Auswertung von PHP-Code veranlasst werden.
  Diese API steht in C-Programmen durch Linken gegen die Bibliothek

[5]Die Abkürzung SAPI steht für Server Application Programming Interface

`php5embed.lib` zur Verfügung. Für die Verwendung bei der Java-Entwicklung ist diese Bibliothek nicht geeignet.

Aus dieser Aufzählung folgt, dass ein spezieller Software-Adapter in Form einer DLL bereitgestellt werden muss. Dieser Software-Adapter muss die folgenden Eigenschaften haben:

- Er kann über das JNI geladen werden und stellt für die Java-Laufzeitumgebung sichtbare Funktionen, bereit.
- er hat Zugriff auf die `embed SAPI` von PHP
- und erledigt Zusatzarbeiten wie Datenypkonversionen zwischen PHP und Java-Datentypen

Damit ergeben sich die in Abbildung 6.4 vereinfacht dargestellten Abhängigkeiten der einzelnen Methoden:

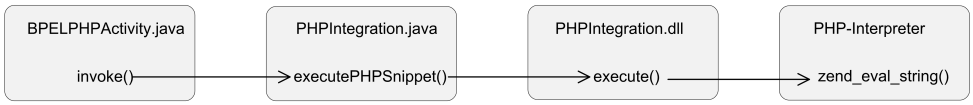

Abbildung 6.4: Detaillierte Abhängigkeiten der Methode invoke()

Die Implementation der Methode `invoke()` wurde bereits vorgestellt. Die Implementationen der Methoden `executePHPSnippet()` sowie der `execute()`-Methode werden in den Abschnitten 6.3.1 beziehungsweise 6.3.3 vorgestellt. Die Implementation des PHP-Interpreters sowie der Funktion `zend_eval_string()`, welche den PHP-Interpreter zum Ausführen von PHP-Code veranlasst, wird durch die PHP-API verborgen und ist für das Verständnis der weiteren Ausführungen nicht relevant.

Mit den bis zu diesem Punkt vorgestellten Maßnahmen ergibt sich das in Abbildung 6.5 dargestellte Zusammenspiel der einzelnen Komponenten, welches im Folgenden beschrieben wird:

Zur Laufzeit des Prozesses soll eine PHP-Snippet-Aktivität ausgeführt werden. Dazu wird die Methode `invoke()` des WPS-Plugins `BPELPHPActivity` aufgerufen, welche die der Aktivität entsprechende Aktion, sprich die Ausführung des PHP-Snippets, durchführen soll. Dazu wird das PHP-Snippet über die in der Abbildung dargestellte Aufruf-Kaskade bis zum PHP-Interpreter durchgereicht. Dieser führt das Snippet aus, wobei lediglich die Seiteneffekte (Veränderung von Variableninformation, Schreiben in Datei o.ä) interessant sind.
Am Ende der Snippet-Ausführung kehrt der PHP-Interpreter ohne Rückgabewert zum Aufrufer zurück und die Auswertung des Geschäftsprozesses wird fortgesetzt.

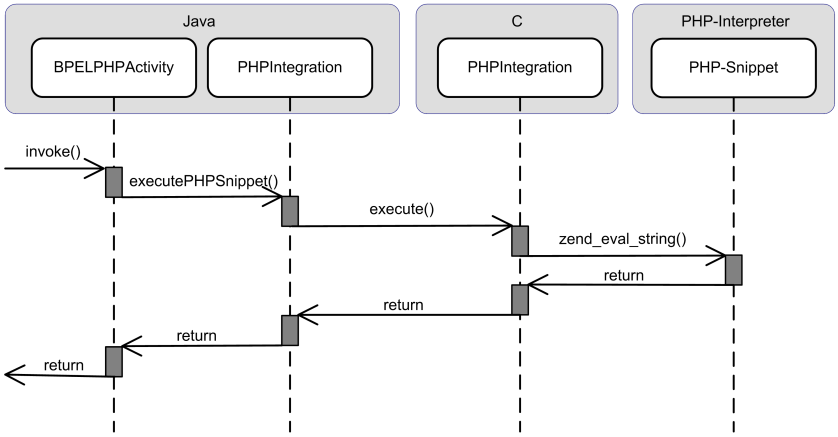

Abbildung 6.5: Sequenzdiagramm der Funktionsaufrufe zum Ausführen von PHP-Code

6.3.1 Verwendung des Java Native Interface

Im Folgenden wird das vollständige Vorgehen zur erfolgreichen Einbindung des PHP-Interpreters über JNI dargestellt. Eine übersichtliche Einführung in die Verwendung des JNI gibt die Arbeit **Using the Java Native Interface** [BS03].
Zur Verwendung einer Funktion, die via JNI importiert wird, genügen in der Java-Quellcodedatei die in Listing 6.5 aufgeführten Befehle.

Listing 6.5: Verwendung einer nativen Methode in Java (Auszug aus PHPIntegration.java)

```
1  static {
2    System.loadLibrary("PHPIntegration");
3  }
4  private native void execute(String snippet, String snippetName, Object callBack);
5  ...
6  execute(snippetCode, snippetName, "a callback");
```

Durch den Befehl `System.loadLibrary("PHPIntegration")` in Zeile 2 wird versucht, die angegebene Systembibliothek zur Laufzeit des Programmes zu laden. Unter dem Betriebssystem Microsoft Windows handelt es sich bei diesen Systembibliotheken grundsätzlich um DLL-Dateien. Damit der Befehl erfolgreich durchgeführt werden kann, muss also eine Bibliothek mit Namen `PHPIntegration.dll` an einer Stelle im Dateisystem des Betriebssystems liegen, an der die Java-Laufzeitumgebung diese auffinden kann[6], ansonsten bricht die Ausführung der Programms an dieser Stelle sofort mit einer Fehlermeldung ab.
In Zeile 4 wird durch das Schlüsselwort `native` die Methode `execute()` als native Methode, die aus einer externen Bibliothek geladen wird, deklariert. Der Java-Compiler

[6]Dies kann unter Windows dadurch erreicht werden, das die dll-Datei in einem Verzeichnis abgelegt wird, welches in der Umgebungsvariable %PATH% aufgelistet ist. Die Java-Laufzeitumgebung sucht sämtliche in %PATH% aufgelistete Verzeichnisse nach einer dll-Datei mit diesem Namen ab.

sucht nicht nach einer entsprechenden Methode im Quelltext der Java-Datei sondern verlässt sich darauf, dass diese Methode zur Laufzeit des Programmes bereitgestellt wird. Der Aufruf der nativen Methode `execute()` in Zeile 6 unterscheidet sich in keiner Weise von einem lokalen Methodenaufruf. Die Tatsache, dass hier eine native Methode aufgerufen wird, bleibt für den Aufrufer an dieser Stelle verborgen.

6.3.2 Erzeugung einer JNI-kompatiblen dynamischen Bibliothek

Zur Laufzeit des Programmes wird versucht, die Methode `execute()` per JNI auszuführen. Damit dies gelingt, muss in der DLL eine Methode gefunden werden können, die sich an die in Listing 6.5, Zeile 4 deklarierte Schnittstelle hält und außerdem einen fest vorgegebenen Namen trägt, welcher sich aus dem Paket- sowie Klassennamen, in dem die Methode deklariert ist, sowie dem Methodennamen selbst zusammensetzt. Der Weg von der Funktionsdeklaration in der Java-Klasse bis zur fertigen Funktion in der DLL wird ausführlich erklärt in dem Buch **The Java Native Interface - Programmers's Guide and Specification** [Lia99]. Dieser Weg ist dargestellt in Abbildung 6.6 und wird im Folgenden erläutert:

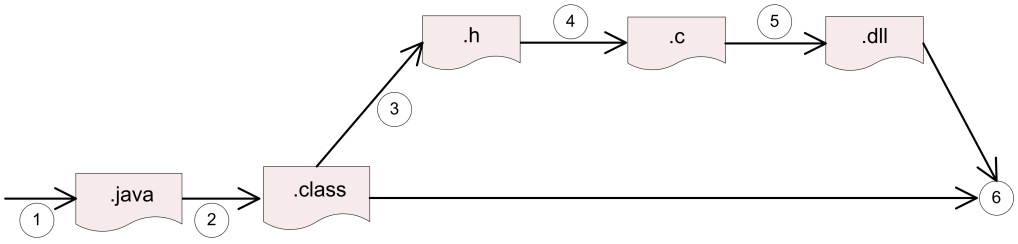

Abbildung 6.6: Erzeugung der DLL aus der Java Quellcodedatei

1. Zuerst wird die Java-Klasse erzeugt, die die native Methode deklariert. Für den konkreten Fall trägt diese Methode den Namen `execute()` und ist deklariert in der Klasse mit dem Namen `invokePHPInterpreter`, welche in dem Paket `com.ibm.bpe.customactivities.php` enthalten ist.
2. Als nächstes wird die Quellcodedatei mittels des Java-Compilers javac in die Klassendatei `invokePHPInterpreter.class` compiliert.
3. Mit Hilfe des Werkzeugs `javah` und der Option `-jni` wird aus dieser Klassendatei automatisch eine C-Header-Datei erzeugt. Diese enthält die Schnittstellendefinitionen sämtlicher zu implementierender C-Routinen samt den korrekten Namen für die Implementationen dieser Funktionen.
4. Ausgehend von der C-Header-Datei kann eine C-Quellcodedatei erstellt werden, die die Implementation der Methode enthält.
5. Diese Quellcodedatei muss mit Hilfe eines C-Compilers in eine DLL compiliert werden. Für das Beispiel in Listing 6.5 muss diese DLL den Namen `PHPIntegration.dll` tragen.
6. Anschließend kann das Java-Programm aufgerufen werden. Zur Laufzeit des Programmes wird sowohl die Java-Klassendefinition

(`invokePHPInterpreter.class`), als auch die dll-Datei (`PHPIntegration.dll`) geladen.

Zu diesen Ausführungen ist Folgendes anzumerken:

- Die beschriebenen Schritte beziehen sich allgemein auf die Verwendung des Java Native Interface. Spezielle Schritte, die nötig sind, um den PHP-Interpreter in die DLL zu integrieren, wurden noch nicht beschrieben. Diese Ausführungen folgen in Abschnitt 6.3.3.
- Die DLL wird für eine bestimmte Plattform Compiliert. Durch deren Verwendung wird die gesamte Java-Anwendung von dieser Plattform abhängig.
- Obwohl die DLL im selben Prozess läuft wie die Java-Laufzeitumgebung, kann die Java-Laufzeitumgebung keine Speicherbereinigung für den von der DLL belegten Speicherbereich durchführen. Das Speichermanagement muss in der DLL selbst vorgenommen werden. Falls dabei Speicherzugriffsfehler auftreten, so ist davon die gesamte Java-Laufzeitumgebung betroffen: Ein Fehler in der DLL kann zum vollständigen Absturz der Java-Laufzeitumgebung und somit des WPS führen. Bei der Programmierung der DLL muss daher sehr vorsichtig vorgegangen werden.

6.3.3 Aufruf des PHP-Interpreters

Zur Erstellung einer DLL, die in der Lage ist PHP-Code auszuführen, sind die folgenden zwei Schritte nötig:

- Die spezielle C-Headerdatei `php_embed.h` muss über ein `include` in das Projekt geladen werden.
- Beim Übersetzen des Projektes muss der Linker die Bibliothek `php5embed.lib` miteinbinden, welche den Zugriff auf die PHP-API zur Verfügung stellt.

Diese beiden Schritte stellen die Grundvoraussetzung zur Verwendung des PHP-Interpreters dar, die einzelnen Schritte, die im Weiteren für die Integration des PHP-Interpreters über JNI nötig sind, sind in Listing 6.6 festgehalten und werden im Folgenden besprochen:

Listing 6.6: Ausführung eines PHP-Snippets über JNI (Auszug aus PHPIntegration.dll)

```
1  #include "sapi\embed\php_embed.h"
2  #include <jni.h>
3  JNIEXPORT void JNICALL Java_com_ibm_bpe_customactivities_php_InvokePHPInterpreter_execute
4  (JNIEnv *env, jobject object, jstring snipCd, ...){
5    ...
6    const char* snippetCode;
7    //copy the jstring containing the snippet code to a c-compatible string
8    snippetCode = env->GetStringUTFChars(snipCd, FALSE);
9
10   //start PHP-Execution-Block
11   PHP_EMBED_START_BLOCK(0, NULL)
12   zend_try {
13     // execute the PHP-Snippet Code
14     int y = zend_eval_string((char*) snippetCode, ...);
```

```
15   } zend_catch {
16     //a serious error (e.g. script parse error) has occured. handle error here
17   } zend_end_try();
18
19   //PHP-Execution End
20   PHP_EMBED_END_BLOCK()
21
22   //free the copy of snippetCode
23   env->ReleaseStringUTFChars(snipCd, snippetCode);
24 ...
25 }
```

Funktionsdeklaration

In Zeile 3 des Listings wird eine Funktion mit dem umständlichen Namen

`Java_com_ibm_bpe_customactivities_php_InvokePHPInterpreter_execute()`

deklariert. Dieser Name entspricht der bereits besprochenen Namenskonvention für JNI-Funktionen und leitet sich wie in Abbildung 6.7 dargestellt aus den folgenden Namensbestandteilen ab:

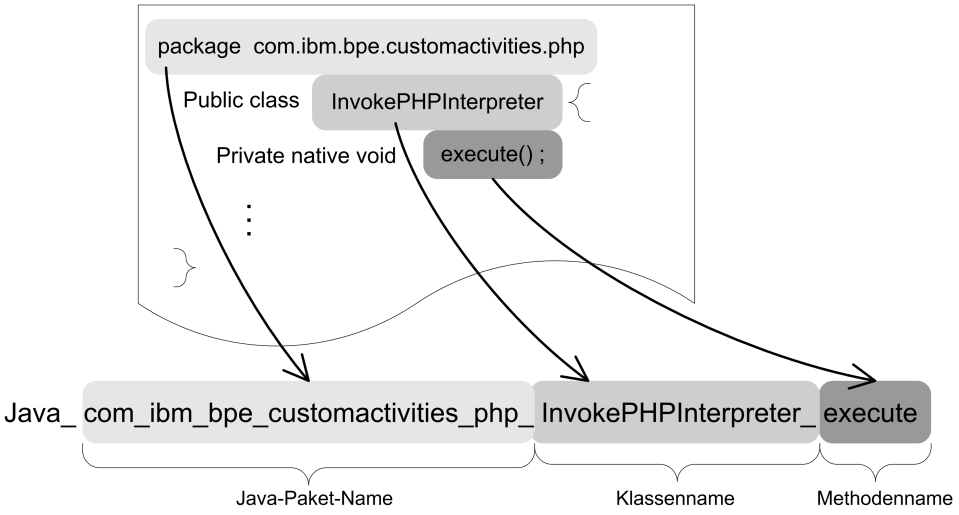

Abbildung 6.7: Bestandteile des Funktionsnamens der nativen Methode

Der Namen der nativen Methode wird grundsätzlich nach folgendem Muster gebildet:

`Java_<packagename>_<classname>_<functionname>`

Das bedeutet für die Entstehung des oben genannten Methodennamens die folgenden Vorgänge: Der nativen Methode wird grundsätzlich die Bezeichnung `Java_` vorangestellt. Es folgen der Name des Pakets, in welchem die Klasse definiert ist (`com.ibm.bpe.customactivities.php`), der Name der Klasse, in welchem die Methode definiert ist (`InvokePHPInterpreter`) sowie der Name der Methode selbst (`execute`). Dabei werden alle Punkte (.) durch Unterstriche (_) ersetzt. Das Ergebnis ist der oben

aufgeführte Methodenname. Der Name der Methode darf nicht geändert werden, da sonst bei der Ausführung des Programmes ein Laufzeitfehler entsteht.

Abgesehen von dem umständlichen Namen enthält die Funktionsdeklaration weitere JNI-spezifische Besonderheiten, diese sind:

- Die beiden JNI-spezifischen Schlüsselwörter `JNIEXPORT` sowie `JNICALL`. Diese sorgen dafür, dass diese Funktion entsprechend den Konventionen für das Java Native Interface generiert werden und so in der DLL abgelegt wird, dass sie beim Aufruf der Methode durch das JNI gefunden werden kann.
- Als zweite Besonderheit besagt die Funktionsdeklaration, dass der Funktion beim Aufruf zwei Parameter übergeben werden, welche in der ursprünglichen Funktionsdeklaration in der Java-Quellcodedatei nicht aufgeführt werden (vgl. Listing 6.5 Zeile 6). Diese beiden Parameter haben die folgende Bedeutung:

  JNIEnv *env ist ein Zeiger auf eine spezielle JNI-spezifische Datenstruktur, über die eine Vielzahl von JNI-API-Aufrufen durchgeführt werden können. Beispiele für solche API-Aufrufe folgen in diesem Abschnitt sowie dem Abschnitt 6.4.

  jobject obj enthält eine Referenz auf das Objekt, welches diesen Funktionsaufruf durchgeführt hat. Mit Hilfe dieser Referenz ist die aufgerufene native Methode in der Lage Rückrufe (*callbacks*) zum Aufrufer durchzuführen. Von dieser Möglichkeit wird in Abschnitt 6.4 Gebrauch gemacht.

Typumwandlung

Datentypen aus Java können nicht direkt auf C-Datentypen abgebildet werden. Java und C verwenden unterschiedliche interne Repräsentationen für diese Typen. Für den Austausch von Daten zwischen Java und C muss daher eine Typumwandlung stattfinden. Diese Typumwandlung ist Teil des JNI und wird im Folgenden beschrieben: Wie aus der Schnittstellenbeschreibung in Zeile 3 des Listings 6.6 ersichtlich ist, wird der Snippetcode in Form eines `jstrings` übergeben. Trotz dieses Typnamens enthält dieser Datentyp keinen String, sondern lediglich eine verborgene Referenz (*opaque reference*)[Lia99], was bedeutet, dass der Datentyp `jstring` einen Zeiger auf interne Datenstrukturen der Java Virtual Machine enthält. Um an die Daten hinter diesem Zeiger kommen zu können, muss die JNI-API-Funktion `GetStringUTFChars()` (Zeile 8) aufgerufen werden, welche über den JNIEnv-Interface-Zeiger bereitgestellt wird. Diese Methode greift über den übergebenen Zeiger auf die Werte, die in der JVM gespeichert sind, zu und erzeugt daraus eine C-kompatible Stringrepräsentation.

Speicherfreigabe

Sobald der mittels der Funktion `GetStringUTFChars()` bereitgestellte String nicht mehr benötigt wird, muss der dafür bereitgestellte Speicher wieder freigegeben werden um ein Speicherleck (*memory leak*) zu vermeiden. Dies wird mittels des Funktionsaufrufes

`ReleaseStringUTFChars()` in Zeile 23 erreicht, mit dem der JVM signalisiert wird, dass der entsprechende Speicherbereich freigegeben werden kann.

Ausführung des Snippetcodes

Die eigentliche Ausführung des PHP-Codes findet in den Zeilen 11 bis 20 statt. Die speziellen Macros `PHP_EMBED_START_BLOCK` respektive `PHP_EMBED_END_BLOCK()` starten beziehungsweise beenden einen PHP-Request. Alle Anweisungen, die innerhalb dieses Request ausgeführt werden, gehören logisch gesehen zusammen und verwenden dieselbe Symboltabelle. Mit anderen Worten, alle Anweisungen in diesem Block entsprechen logisch gesehen der Ausführung eines PHP-Skriptes, wie man es typischerweise aus der Webanwendungs-Entwicklung her kennt.
Der zentrale Aufruf der zur Auswertung des PHP-Codes verwendet wird, ist der Aufruf der Funktion `zend_eval_string()` in Zeile 14. Mittels dieses Aufrufes wird das beim Aufruf der Funktion übergebene PHP-Snippet vollständig ausgeführt. Die Ausführung des PHP-Snippets ist in einen speziellen `try-catch`-Block eingebettet, mit dessen Hilfe im Fehlerfall gesonderte Aktionen ausgeführt werden können.

6.3.4 Umleiten der Standardausgabe

Bei WPS werden die Standardausgabe sowie die Standardfehlerausgabe auf spezielle Dateien umgeleitet: Die Standardausgabe wird auf die Datei `SystemOut.log` umgeleitet, während die Standardfehlerausgabe auf `SystemErr.log` umgeleitet wird. Dieses Vorgehen ist durchaus sinnig, da WPS nicht interaktiv betrieben wird und für gewöhnlich die Ausgaben lediglich zu Dokumentationszwecken gedacht sind. Während der Entwicklung eines Geschäftsprozesses in WID werden diese beiden Ausgabedateien von WID überwacht und innerhalb eines Konsolenfensters in WID angezeigt. Auf diese Weise kann der Entwickler bequem prüfen, ob die Ausgaben des Prozesses korrekt sind, ohne sich mit den einzelnen Log-Dateien des Servers auseinanderzusetzen.
Für die PHP-Snippets ergibt sich an dieser Stelle das folgende Problem: Die Ausgaben, die durch native-calls verursacht werden, werden von WPS in separate Dateien geschrieben, die die Namen `native_stdout.log` beziehungsweise `native_stderr.log` tragen. Diese Dateien werden von WID jedoch nicht überwacht, weshalb die Ausgaben des PHP-Snippets sowie eventuelle Fehlermeldungen nicht in der Konsole von WID erscheinen.
Wie dieser Umstand umgangen werden kann, wird für das Beispiel der Standardausgabe in Listing 6.7 gezeigt. Für die Standardfehlerausgabe wurde ein äquivalentes Verfahren vorgesehen.

Listing 6.7: Umleiten der PHP-Standardausgabe (Auszug aus PHPIntegration.dll)

```
1 /*
2  * redirects the string provided by str to the java SysOut-Method
3  */
4 static int embed4_ub_write(const char *str, unsigned int str_length TSRMLS_DC){
5   env->CallVoidMethod(obj, sysOut, env->NewStringUTF(str));
6   return 0;
```

```
7  }
8  // override the standard output methods with new ones.
9  php_embed_module.ub_write = embed4_ub_write;
10 ...
11 PHP_EMBED_START_BLOCK()
12 ...
```

Für das Verständnis des Listings 6.7 sind folgende Sachverhalte von Bedeutung:

- Sämtliche Ausgaben, die in PHP-Skripten mit Ausgabefunktionen wie beispielsweise `echo()` oder `printf()` erzeugt werden, werden PHP-intern über ein- und dieselbe Funktion realisiert: Die Funktion `ub_write()`.
- Mit Hilfe des speziellen Konstruktes `php_embed_module` können eine Vielzahl von PHP-Einstellungen und Standards vor der eigentlichen PHP-Ausführung überschrieben werden. Mit Hilfe dieses Konstruktes kann beispielsweise die Funktion, die zur Ausgabe von Daten auf die Standardausgabe verwendet wird, überschrieben werden.

Die einzelnen Schritte, die im Quellcode-Ausschnitt aus der DLL vorgestellt werden, sind demnach:

- Zeile 4: Deklaration einer Funktion mit Namen `embed4_ub_write()`, die die selbe Schnittstelle hat, wie die Funktion `ub_write()`.
- Zeile 9: Überschreiben der Funktion `ub_write()` mit der neuen Funktion `embed4_ub_write()`.

Das Macro TSRMLS_DC in der Schnittstellenbeschreibung hängt mit speziellen threading-Eigenschaften des PHP-Interpreters zusammen und ist für die aktuelle Betrachtung unerheblich[7].
Die Implementation der neuen Standard-Ausgabe-Funktion umfasst gerade einmal eine Zeile (5). Diese Zeile enthält einen Callback zu einer Funktion des Aufrufers, welche mit `sysOut` bezeichnet ist. Die Java-Methode `sysOut` schreibt den übergebenen Parameter auf die Standard-Ausgabe von Java. Auf diese Weise wird erreicht, dass die Ausgaben des PHP-Snippets grundsätzlich an der selben Stelle erscheinen, wie die Ausgaben der Java-Snippets. Die Details der Callback-Funktionen werden im folgenden Abschnitt über die Erweiterung von PHP im Detail erläutert.

6.4 PHP-Erweiterung

Im vorausgegangenen Abschnitt wurden die wichtigen Arbeitsschritte zur Integration des PHP-Interpreters in die JVM beschrieben. Zusammen mit den in Abschnitt 6.1 beschriebenen Maßnahmen ist somit die Integration von PHP in BPEL und die Ausführung von PHP durch WPS gewährleistet. Allerdings fehlt der in Abschnitt 5.2 geforderte lesende und schreibende Zugriff auf die im Geschäftsprozess definierten

[7] Die Details dazu können beispielsweise einem Artikel von Sara Golemon auf http://blog.libssh2.org/index.php?/archives/22-What-the-heck-is-TSRMLS_CC-anyway.html entnommen werden.

Variablen. Wie in Abschnitt 5.2.2 festgestellt wurde, ist ein solcher Zugriff über eine bidirektionale Kommunikation mit Direktzugriff möglich.
Eine solche Kommunikation muss über eine Erweiterung für die PHP-Laufzeitumgebung realisiert werden. Die wichtigen Schritte zur Erstellung dieser Erweiterung werden im Folgenden beschrieben.

Die Umsetzung der PHP-Erweiterung erfordert im Wesentlichen die Beschäftigung mit zwei separaten Teilaspekten:

- Der Zugriff auf Variableninformationen des Geschäftsprozesses muss ermöglicht werden.
- Der Zugriff auf Variablen mit komplexem Datentyp soll in PHP-Snippets über die SDO-API erfolgen.

Der Zugriff auf Variableninformationen wird über die im Rahmen dieser Arbeit entwickelte PHP-Erweiterung `PHPIntegration` bereitgestellt. Die Entwicklung dieser Erweiterung wird in Abschnitt 6.4.3 vorgestellt.

Die SDO-API wird von der in Abschnitt 6.4.4 vorgestellten PHP-Klasse `SDO_DAS_BPC` bereitgestellt. Diese Klasse kombiniert Funktionalität, die durch die Erweiterungen `PHPIntegration` sowie `php_sdo` bereitgestellt werden.

6.4.1 Beteiligte Komponenten

Bevor auf die Details der einzelnen Komponenten der PHP-Erweiterung eingegangen wird, wird hier eine Übersicht über die einzelnen Komponenten, sowie deren Zuständigkeiten und Abhängigkeiten gegeben.

Die Erweiterung für PHP setzt sich aus den folgenden drei Bestandteilen zusammen:

- **Java-Callback-Funktionen**: Der Zugriff auf die Variableninformation wird über spezielle Callback-Funktionen, welche Zugriff auf den Prozesskontext haben, realisiert. Diese Callback-Funktionen sind in Java implementiert und werden über JNI angesprochen. Sie werden in Abschnitt 6.4.2 vorgestellt.
- **C-Adapter**: Um auf die angesprochenen Callback-Funktionen aus PHP heraus zugreifen zu können, ist eine in der Programmiersprache C entwickelte DLL nötig, welche die Callback-Funktionen in einer PHP-kompatiblen Weise zur Verfügung stellt. Die nötigen Schritte zur Entwicklung dieser DLL werden in Abschnitt 6.4.3 dargestellt.
- **PHP-Anteil**: Über das reine Aufrufen von Callback-Funktionen hinaus ist teilweise die Durchführung von zusätzlichen Arbeitsschritten nötig. Um die Details der Implementierung soweit wie möglich transparent für den Anwender zu halten, wird diese Zusatzarbeit durch eine separate Klasse – die PHP-Klasse `SDO_DAS_BPC` – vorgenommen. Diese Klasse wird in Abschnitt 6.4.4 beschrieben.

Zuständigkeiten und Abhängigkeiten der einzelnen Komponenten

Im Folgenden wird beschrieben, welche Zuständigkeiten die einzelnen Komponenten haben und wie diese voneinander abhängen, um den Zugriff auf die Prozessvariablen zu ermöglichen. Die Abhängigkeiten der einzelnen Komponenten beim Arbeiten mit Variableninformation ist dargestellt in Abbildung 6.8.

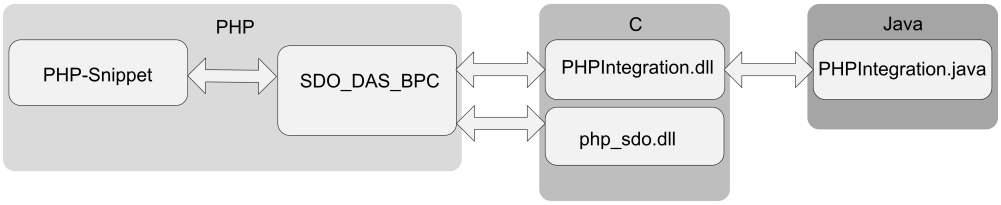

Abbildung 6.8: Abhängigkeiten der einzelnen Komponenten der PHP-Erweiterung

Zuständigkeit und Abhängigkeit der Klasse SDO_DAS_BPC
Für alle Aktionen, die sich mit Prozessvariablen befassen, kann in dem PHP-Snippet auf die spezielle Klasse `SDO_DAS_BPC` zugegriffen werden. Diese Klasse bietet den Zugriff auf sämtliche Variablen im Prozesskontext, sowie weitere Funktionalität, die für das Arbeiten mit Prozessvariablen hilfreich ist. Dabei hängt sie von den folgenden zwei Komponenten ab:

- Um Informationen über die Variablen aus dem Prozesskontext abzufragen, bedient sie sich der Funktionalität der PHP-Erweiterung `PHPIntegration`.
- Um den Zugriff auf Variablen mit komplexem Datentyp über die SDO-API bereitstellen zu können, greift sie auf Funktionalität zurück, welche von der SDO-Erweiterung `php_sdo` geliefert wird.

Zuständigkeit und Abhängigkeit der Erweiterung PHPIntegration.dll
Die Bibliothek `PHPIntegration.dll` stellt das Bindeglied zwischen dem PHP-Kontext und dem Java-Kontext her. Sie bietet für den PHP-Interpreter sichtbare Funktionen über den PHP-Erweiterungsmechanismus an. Diese Funktionen führen eine geeignete Konvertierung der Eingabeparameter durch und rufen anschließend über das JNI Funktionen der Java-Klasse `PHPIntegration.java` auf. Abgesehen von der Abhängigkeit auf spezielle PHP-API-Funktionen, welche im Schaubild der Übersicht halber nicht eingezeichnet wurden, hängt die Bibliothek `PHPIntegration.dll` also im Wesentlichen von der Java-Klasse `PHPIntegration.java` ab.

Zuständigkeit und Abhängigkeit der Erweiterung php_sdo.dll
Die Erweiterung `php_sdo.dll` stellt das Arbeiten auf komplexen Datentypen mittels der SDO-API[8] zur Verfügung. Für die korrekte Funktionsweise dieser Erweiterung ist PHP in einer Version ab 5.1.0 sowie die Erweiterung `libxml2` nötig, welche in PHP standardmäßig aktiviert ist.

[8] Details des SDO-Konzepts werden in Abschnitt 3.6.4 vorgestellt.

Zuständigkeit und Abhängigkeit der Java-Klasse PHPIntegration.java
Die Funktionen der Java-Klasse haben Zugriff auf den Prozesskontext und führen die angeforderten Aktionen auf den Prozessvariablen durch. Die Ergebnisse der Methodenaufrufe werden auf dem gleichen Weg wie die Methodenaufrufe zurück gereicht, allerdings in umgekehrter Reihenfolge. Für die Erbringung der Funktionalität ist die Klasse `PHPIntegration.java` vom Zugriff auf die Prozessvariablen sowie einigen Hilfsfunktionen abhängig.

Einige Details dieser Komponenten werden im Folgenden beschrieben:

6.4.2 Bereitstellen der Callback-Funktionen

Zum Zugriff auf die Variableninformationen der laufenden Prozessinstanz kann innerhalb des WPS-Plugins auf spezielle Java-API-Aufrufe zurückgegriffen werden, mit deren Hilfe der gleiche Funktionsumfang wie in den Java-Snippets realisiert werden kann. Aufbauend auf dieser API werden durch die Java-Klasse `PHPIntegration` die folgenden, in Abbildung 6.9 dargestellten Funktionen bereitgestellt.

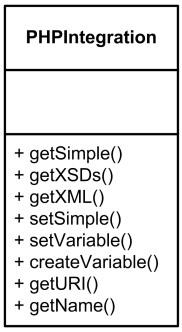

Abbildung 6.9: Callback-Funktionen der Java-Klasse PHPIntegration

Die Namen dieser Funktionen sind selbstbeschreibend, daher wird an dieser Stelle auf die Beschreibung der angebotenen Funktionalität verzichtet. Wozu die Funktionen `getXML()` sowie `getXSDs()` benötigt werden, wird in Abschnitt 6.4.4 geklärt.
Um diese Funktionen über JNI aufrufbar zu machen, werden im Java-Code keine speziellen Auszeichnungen benötigt. Allerdings müssen die Funktionen selbstverständlich als `public` deklariert sein.

6.4.3 Erzeugung der PHP-Erweiterung

Über die oben vorgestellten Java-Funktionen werden Informationen über die Prozessvariablen über das JNI zugreifbar. Um auf diese Funktionen von PHP aus zugreifen zu können, muss eine Erweiterung für PHP bereitgestellt werden, die über den Standard-Erweiterungsmechanismus von PHP geladen werden kann und gleichzeitig über das JNI auf die speziellen Callback-Funktionen zugreifen kann.

Wie bereits dargestellt wurde, kann eine PHP-Erweiterung in Form einer DLL bereitgestellt werden. Die wichtigen Schritte zur Entwicklung dieser DLL werden im Folgenden angesprochen. Für ausführlichere Anleitungen zum Erstellen eigener PHP-Erweiterungen sei auf die Bücher **Extending und Embedding PHP** [Gol06] sowie **Advanced PHP Programming** [Sch04] verwiesen.

Eigenschaften der Erweiterung festlegen

In der C-Header-Datei der Erweiterung können einige generelle Eigenschaften der Erweiterung festgelegt werden. Listing 6.8 zeigt einige dieser generellen Eigenschaften. Diese sind im Einzelnen:

Listing 6.8: Eigenschaften der Erweiterung festlegen (Auszug aus PHPIntegration.h)

```
1  /*Define Extension Properties*/
2  #define PHPIntegration_VERSION "1.0"
3  #define PHPIntegration_EXTNAME "PHPIntegration"
4
5  /*declare the Functions available through this extension*/
6
7  PHP_FUNCTION(nGetUri);
8  PHP_FUNCTION(nGetName);
9  PHP_FUNCTION(nGetSimple);
10 /* more functions provided by the extensions follow here */
11
12 /*Define the entry point symbol
13   Zend will use when loading this extension*/
14
15 extern zend_module_entry PHPIntegration_module_entry;
16 #define phpext_PHPIntegration_ptr &PHPIntegration_module_entry
```

- **Versionsname sowie Versionsnummer** sowie weitere Eigenschaften der PHP-Erweiterung können über C-übliche `define`-Statements festgelegt werden (siehe Zeilen 2+3).
- **Funktionen der Erweiterung** werden mittels des speziellen PHP-Macros `PHP_FUNCTION` (Zeilen 7 ff) angemeldet. Auf diese Weise können sie in die Funktionstabelle aufgenommen werden, welche der PHP-Interpreter beim Laden der Erweiterung einliest.
- **Einsprungspunkt festlegen**: Wie schon JNI, so erwartet auch PHP ein bestimmtes Format der DLL. Während bei JNI die angebotenen Funktionen einem strikten Namenschema folgen müssen, um aufgefunden zu werden, erwartet PHP eine spezielle Datenstruktur in der DLL zu finden, in der sich Zeiger auf sämtliche angebotenen Funktionen befinden. Diese Datenstruktur wird in der Header-Datei angemeldet und in der im Folgenden vorgestellten C-Datei mit Einträgen gefüllt.

Implementation der Erweiterungsfunktionen

Bei der Bibliothek `PHPIntegration.dll` handelt es sich lediglich um einen Software-Adapter, das heißt um ein Verbindungsstück, das die Kommunikation zwischen Java

und PHP möglich macht. Sämtliche Funktionen in der DLL reichen ihre Aufrufe im Wesentlichen – abgesehen von Typkonversionen oder Ähnlichem – lediglich weiter an Funktionsaufrufe des PHP-Interpreters oder Funktionen des WPS-Plugins. Aus diesem Grund haben sämtliche PHP-Erweiterungsfunktionen einen relativ ähnlichen Aufbau. Dieser Aufbau wird im Folgenden anhand der PHP-Erweiterungsfunktion `nGetSimple()` gezeigt, mit der Variableninformationen über eine simpel getypte Variable abgefragt werden können.

Listing 6.9: Implementation einer PHP-Erweiterungsfunktion (Auszug aus PHPIntegration.c)

```
1  /*
2   * Obtains the Contents of a simple typed Variable
3   * input: string varName
4   * output: string VariableContent
5   */
6
7  PHP_FUNCTION(nGetSimple){
8    char * varName;
9    int varNameLength;
10
11   if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "s", &varName, &varNameLength) ==
         FAILURE) {
12     RETURN_NULL();
13   }
14
15   const char * str;
16   jstring jstr;
17   jstr = (jstring) env->CallObjectMethod(obj, getSimpleVar, env->NewStringUTF(varName));
18
19   str = env->GetStringUTFChars(jstr, FALSE);
20   if (str == NULL) {
21     return; /* out of memory */
22   }
23   char* variableContent = estrdup(str);
24   env->ReleaseStringUTFChars(jstr, str);
25   RETURN_STRING(variableContent, 0);
26 }
```

Dieses Listing enthält abgesehen von Fehlerbehandlung und Datentypkonversionen die folgenden wichtigen Bestandteile:

- Mittels des speziellen PHP-Macros `PHP_FUNCTION` (Zeile 7) wird diese Funktion als Funktion markiert, welche aus der DLL exportiert und PHP zur Verfügung gestellt wird.
- Über den speziellen API-Aufruf `zend_parse_parameters()` (Zeile 11), werden die Parameter, die der PHP-Funktion beim Aufruf übergeben wurden, eingelesen. In diesem Fall wird der Name der Variable, deren Inhalt abgefragt werden soll, in die Variable `varName` eingelesen.
- In Zeile 17 wird schließlich der Aufruf der Java-Callback-Funktion durchgeführt. Mittels des Aufrufes `env->CallObjectMethod()` wird über das JNI auf die Java-Funktion `getSimpleVar()` zugegriffen. Diese Funktion erwartet gemäß ihrer Signatur[9] exakt einen Parameter, nämlich den Namen der Variablen, deren Inhalt

[9] Die vollständige Signatur dieser Funktion im Java-Quelltext lautet:

bereitgestellt werden soll. Dieser wird durch die C-Variable `varName` gehalten, welche durch das zuvor beschriebene `zend_parse_parameters()` entsprechend belegt wurde.

6.4.4 Die PHP-Klasse SDO_DAS_BPC

In den vorausgegangenen Ausführungen wurden die ersten beiden Bestandteile der PHP-Erweiterung, nämlich die Java-Callback-Funktionen sowie die C-Adapter-Funktionen beschrieben. In diesem Abschnitt wird der PHP-Anteil der PHP-Erweiterung beschrieben, welcher in der Klasse `SDO_DAS_BPC` implementiert wurde.

Entwicklung des SDO_DAS_BPC
Wie in Abbildung 6.8 bereits angedeutet wurde, erfolgt sämtlicher Datenzugriff aus dem PHP-Snippet über die PHP-Klasse `SDO_DAS_BPC`. Diese hat im wesentlichen die Aufgabe, das Arbeiten auf komplexen Datentypen mittels der SDO-API zu ermöglichen. Gleichzeitig dient sie auch dazu, Implementierungsdetails vor dem PHP-Snippet-Programmierer zu verbergen. Die Klasse `SDO_DAS_BPC` bietet dem PHP-Snippet die in Abbildung 6.10 dargestellten Funktionen:

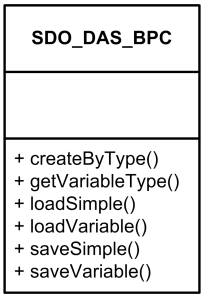

Abbildung 6.10: Funktionalität des SDO_DAS_BPC

Die Namen der Funktionen sind selbsterklärend, ihre Zuständigkeiten werden daher hier nicht beschrieben. Für Details der Funktionsweise wird auf die Implementierung der Funktionen verwiesen, welche der beigelegten CD entnommen werden kann. Die beiden angesprochenen Eigenschaften **Verbergung von Implementationsdetails** sowie **Bereitstellung der SDO-API** werden im Folgenden erläutert:

Verbergung von Implementationsdetails

Bei einigen der in Abbildung 6.10 dargestellten PHP-Funktionen handelt es sich lediglich um Stub-Funktionen, das heißt um Funktionen, die ohne jegliche Zusatzfunktion eine Funktion in der C-Erweiterung aufrufen. Eine dieser Funktionen ist beispielsweise die Funktion `loadSimple()`, deren Implementation in Listing 6.10 abgedruckt ist:

Solch eine Funktion ist im Grunde genommen überflüssig, da stattdessen in den PHP-Snippets auch direkt auf die Funktion `nGetSimple()` zugegriffen werden kann, wie es

```
public String getSimpleVar(String varName)
```

Listing 6.10: Implementation der Funktion loadSimple (Auszug aus SDO_DAS_BPC)

```
1  public function loadSimple($varName){
2     return nGetSimple($varName);
3  }
```

in Abbildung 6.11 dargestellt ist. Der Grund dafür, dass diese Funktion in die Klasse `SDO_DAS_BPC` aufgenommen wurde ist, dass dadurch sämtliche Funktionen, mit denen im PHP-Snippet auf Prozessvariablen zugegriffen wird, über die Klasse `SDO_DAS_BPC` verfügbar sind.

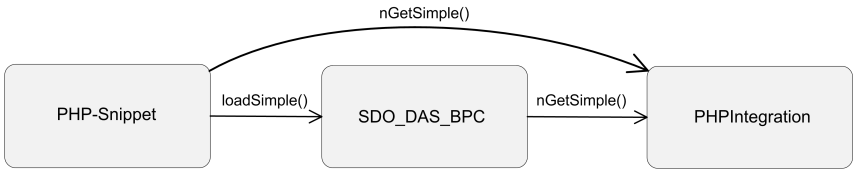

Abbildung 6.11: Direkter Zugriff auf die Funktion nGetSimple()

Selbstverständlich ist der direkte Zugriff auf die Funktion `nGetSimple()` aus dem PHP-Snippet heraus möglich, jedoch ist sie durch diese Weiterleitung nicht mehr nötig. Dies reduziert die Komplexität der PHP-Erweiterung aus Sicht des PHP-Snippet-Programmierers, da er über die Klasse `SDO_DAS_BPC` auf die volle API, die über die DLL bereitgestellt wird, zugreifen kann.

Der eigentliche Nutzen der Klasse `SDO_DAS_BPC` besteht dagegen in der Bereitstellung der SDO-API, was im Folgenden beschrieben wird:

Bereitstellung der SDO-API

Um auf komplexen Datentypen mittels der SDO-API arbeiten zu können, kann auf die existierende SDO-Erweiterung für PHP zurückgegriffen werden[10]. Diese wird in PHP-Skripten durch das Laden der Bibliothek `php_sdo.dll` verfügbar.
Wie bereits in Abschnitt 3.6.4 auf Seite 29 beschrieben wurde, wird für das Laden der Daten ein so genannter Data Mediator Service (DMS) oder auch Data Access Service (DAS) verwendet. Die SDO-Erweiterung für PHP beinhaltet die folgenden Data Access Services:

- Mittels des **SDO_DAS_XML** kann auf Datenquellen im XML-Format zugegriffen werden.
- Der **SDO_DAS_Relational** dagegen bietet den Zugriff auf Daten, welche in einer relationalen Datenbank gespeichert sind.

[10] Die Beschreibung des Funktionsumfangs der SDO-Erweiterung kann Online unter `http://uk2.php.net/sdo` (zuletzt besucht 28.04.2007) abgerufen werden.

Wie aus der Auflistung zu entnehmen ist, enthält diese SDO-Erweiterung keinen DAS für den Zugriff auf Daten, welche in einer laufenden Prozessinstanz eines Geschäftsprozesses definiert sind. Um die existierende SDO-Erweiterung in den Snippets verwenden zu können, ist somit die Entwicklung eines neuen DAS nötig, welcher in Anlehnung an das Bezeichnungsschema der existierenden DASs mit `SDO_DAS_BPC`[11] bezeichnet wurde.

Da sowohl in PHP als auch in Java Bibliotheken zum Umgang mit XML-Dateien existieren, bietet es sich an, für die Entwicklung des `SDO_DAS_BPC` den bereits existierenden `SDO_DAS_XML` zu verwenden. Dazu sind die folgenden Schritte nötig:

- Für die Erzeugung des `SDO_DAS_XML` werden die XML Schema Definitionen des jeweiligen Datentyps, mit dem gearbeitet werden soll benötigt. Diese Information über die Variable muss über einen entsprechenden API-Aufruf bereitgestellt werden.
- Die in der Variablen gespeicherte Information kann über den API-Aufruf `loadString()` des `SDO_DAS_XML` geladen werden. Dazu ist eine XML-Repräsentation des Variableninhalts der angeforderten Variable nötig. Diese Information muss ebenfalls über einen entsprechenden API-Aufruf beschafft werden.
- Das auf diese Weise entstandene Objekt kann an den Aufrufer zurückgeliefert werden. Die Tatsache, dass der `SDO_DAS_XML` verwendet wurde, bleibt für den Aufrufer verborgen.

Der interne Ablauf beim Laden von Variablen mit komplexem Typ ist in Abbildung 6.12 dargestellt.

Der Vollständigkeit halber wurden in dieser Abbildung die Funktionsaufrufe zum Ausführen von PHP-Code integriert, diese sind bereits aus Abbildung 6.5 bekannt. Die Funktionsaufrufe, die mit dem Laden von Variableninformation zusammenhängen sind in der Abbildung grau unterlegt und werden im Folgenden erläutert:
Zum Laden von Variablen mit komplexem Typ (Business Objects) kann in den PHP-Snippets die Funktion `loadVariable()` verwendet werden. Diese liefert ein Business Object zurück, auf dem mittels der SDO-API für PHP gearbeitet werden kann. Wie aus der Abbildung ersichtlich ist, greift die Funktion `loadVariable()` zu diesem Zweck auf die Callback-Funktionen `getXSDs()` sowie `getXML()` zurück. Mit Hilfe der so gewonnenen Information erzeugt sie zunächst einen `SDO_DAS_XML` und lädt daraufhin die in XML serialisierte Variableninformation. Das entstandene Business Object wird anschließend an das aufrufende PHP-Snippet zurückgegeben.

[11] Zur Erinnerung: Die zentrale Komponente des WPS, welche mit der Ausführung von Geschäftsprozessen betraut ist, ist der Business Process Choreographer (BPC).

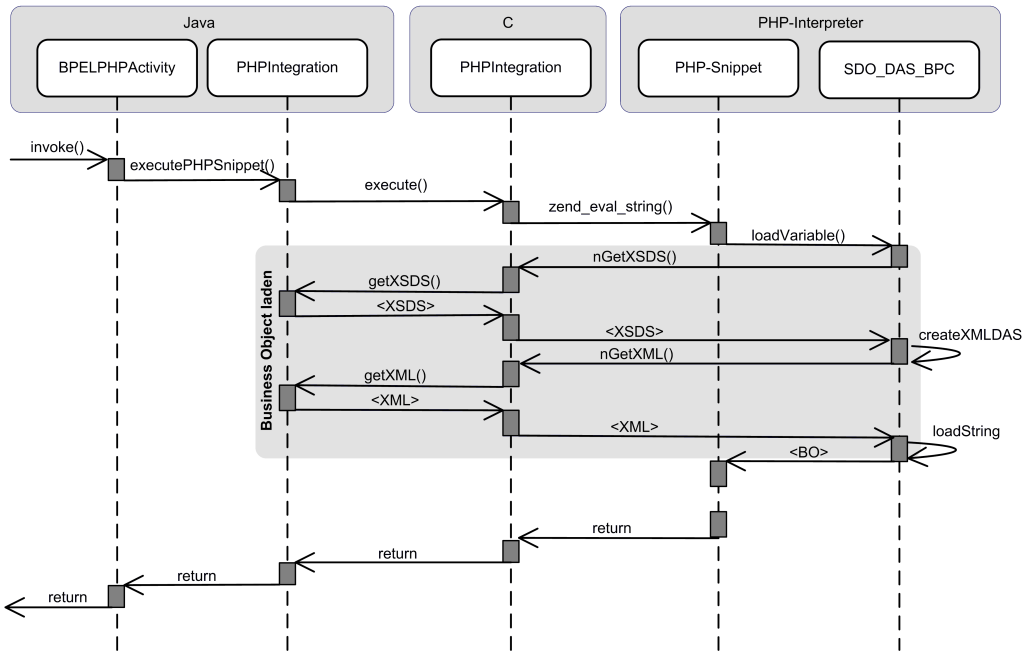

Abbildung 6.12: Laden einer Variable mit komplexem Datentyp

7 Ergebnisse

Im Rahmen dieser Arbeit wurde eine Erweiterung zum BPEL-Standard definiert, mit welcher die Integration von Skriptsprachen in BPEL möglich ist. Es wurde ein Plugin für WPS entwickelt, welches BPEL-Prozesse, die diese Erweiterung verwenden, erkennt und korrekt behandelt. Des Weiteren wurden verschiedene Möglichkeiten untersucht, die Ausführung von PHP-Code für WPS möglich zu machen. Aus den vorgestellten Möglichkeiten erwies sich die Umsetzung mittels Java Native Interface als beste Lösung. Sämtliche Komponenten, die in Kapitel 6 über die Implementierung vorgestellt wurden, wurden im Rahmen dieser Arbeit implementiert und erfolgreich getestet. Abbildung 7.1 gibt einen Gesamtübersicht über die an dieser Lösung beteiligten Komponenten.

Aus dieser Abbildung ist zu erkennen, dass WPS über ein Plugin erweitert wurde. Dieses Plugin stellt über das JNI und eine spezielle, im Rahmen dieser Arbeit entwickelte DLL den Kontakt zum PHP-Interpreter her. Auf diese Weise ist die Ausführung von PHP-Code möglich.
In einem zweiten Arbeitsschritt wurde der Zugriff auf Prozessvariablen einer laufenden Prozessinstanz möglich gemacht. Dazu wurde die bestehende DLL erweitert, so dass sie als PHP-Erweiterung geladen werden kann. Dieser zweite Teil der DLL stellt – wiederum über das JNI – Kontakt zu speziellen Callback-Funktionen des WPS-Plugins her. Auf diese Weise wird in den PHP-Snippets der Zugriff auf die Prozessvariablen ermöglicht. Dabei wurde versucht, die Arbeit mit den Prozessvariablen in den PHP-Snippets ähnlich komfortabel wie in den Java-Snippets zu realisieren. Details dazu werden im Folgenden vorgestellt.

Arbeiten mit den Prozessvariablen

Im Gegensatz zu den Java-Snippets ist es in PHP-Snippets nicht möglich, Daten ohne explizites Laden zu lesen beziehungsweise ohne explizites Speichern in den Prozess-

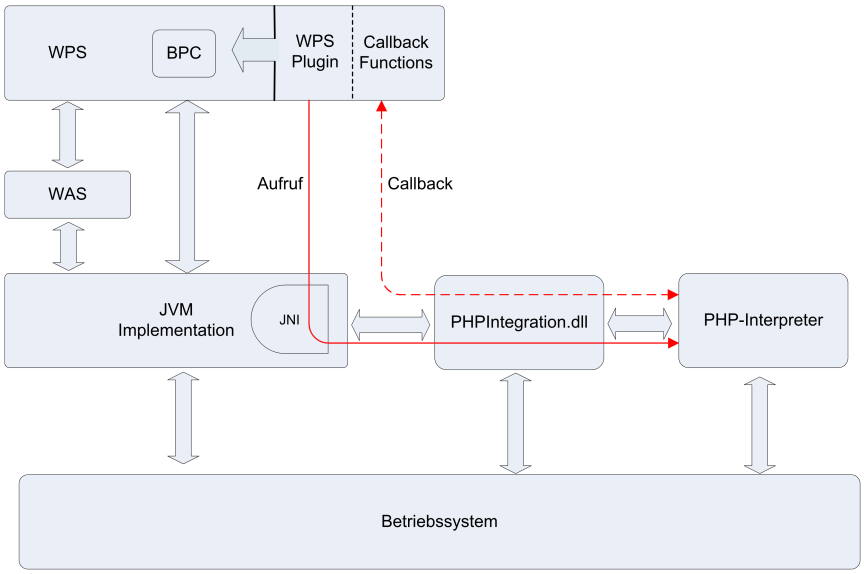

Abbildung 7.1: Erweiterung von PHP mittels JNI

kontext zu schreiben: Im Gegensatz zu den Java-Snippets laufen die PHP-Snippets nicht in der Java-Laufzeitumgebung, sondern im PHP-Interpreter ab und haben somit keinen direkten Zugriff auf die Prozessvariablen. Der Zugriff auf die Daten des Prozesses wird daher über eine gesonderte Kommunikation geregelt, welche explizites Laden und Schreiben von Variablen erfordert.

Der Zugriff auf die Prozessvariablen in PHP orientiert sich direkt an der für die Java-Snippets gewählten Zugriffsweise, welche in Abschnitt 3.2.3 vorgestellt wurde. Im Folgenden wird beschrieben, wie in PHP-Snippets auf die unterschiedlichen Variablentypen zugegriffen werden kann.

Arbeiten mit Variablen mit simplem Datentyp

Ein zu dem Java-Snippet in Listing 3.5 (siehe Seite 29) äquivalentes PHP-Snippet hat den in Listing 7.1 abgedruckten Aufbau:

Listing 7.1: Zugriff auf eine Variable mit simplem Typ in einem PHP-Snippet

```
1  //lesender Zugriff auf eine Variable mit simplem Typ
2  $simpleString = SDO_DAS_BPC::loadSimple("simpleString");
3  echo($simpleString);
4  //schreibender Zugriff auf eine Variable mit simplem Typ
5  $simpleString = "neue Wertbelegung";
6  SDO_DAS_BPC::saveSimple("simpleString", $simpleString);
```

Dazu ist Folgendes anzumerken:

- Die Klasse `SDO_DAS_BPC` enthält den PHP-seitigen Teil der PHP-Erweiterung.

Über diese Klasse wird die API der PHP-Erweiterung in den PHP-Skripten verfügbar gemacht. Diese Klasse wurde in Abschnitt 6.4.4 genauer vorgestellt.

- Das explizite Laden von Variablen mit simplem Datentyp geschieht mittels des Aufrufes `loadSimple()` unter Angabe des Namens der Variable, welche geladen werden soll. Als Rückgabe liefert dieser Aufruf den Wert der Variablen, welcher in Zeile 2 des Listings in der lokalen PHP-Variable `$simpleString` abgespeichert wird.
- Das explizite Speichern von Variablen mit simplem Datentyp geschieht mittels des Aufrufes `saveSimple()` unter Angabe des Namens der Variable, in welche die Information gespeichert werden soll, sowie der neuen Wertbelegung für die Variable. In dem Beispiel wird die Prozessvariable mit dem Namen `simpleString` mit dem Wert der PHP-Variablen `$simpleString` belegt.

Arbeiten mit Variablen mit komplexem Datentyp

Der Zugriff auf komplexe Datentypen wird in Java-Snippets über die Service-Data-Object-API realisiert. Für PHP existiert eine experimentelle Erweiterung für Service Data Objects, welche für den Zugriff auf die Prozessvariablen verwendet werden kann. Durch die Verwendung der SDO-Erweiterung für PHP kann die Arbeit mit den Prozessvariablen in PHP auf ähnliche Weise geschehen, wie dies in Java-Snippets möglich ist.

Dies wird durch den PHP-Code in Listing 7.2 verdeutlicht:

Listing 7.2: Zugriff auf eine Variable mit komplexem Typ in einem PHP-Snippet

```
1  //lesender Zugriff auf eine Variable mit komplexem Typ
2  $dataHandle = SDO_DAS_BPC::loadVariable('kunde1');
3  $kunde1 = $dataHandle->getRootDataObject();
4  echo($kunde1->Vorname ." ". $kunde1->Nachname);
5  //schreibender Zugriff auf eine Variable mit komplexem Typ
6  $kunde1->Vorname = "Max";
7  $kunde1->Nachname = "Mustermann";
8  SDO_DAS_BPC::saveVariable('kunde1', $dataHandle);
```

Dazu ist Folgendes anzumerken:

- Auch für den Zugriff auf komplexe Datentypen ist in den PHP-Snippets explizites Laden und Speichern nötig. Dies geschieht für komplexe Datentypen über die Aufrufe `loadVariable()` in Zeile 2 respektive `saveVariable()` in Zeile 8. Sämtliche Änderungen an der lokalen Kopie der Daten werden im Prozesskontext erst nach der Durchführung des Kommandos `saveVariable()` sichtbar.
- Der Aufruf der Funktion `loadVariable()` liefert ein PHP-Objekt vom Typ `SDO_DAS_XML_Document` zurück. Mittels der Methode `getRootDataObject()` in Zeile 3 wird das eigentliche Business Object extrahiert. Auf diesem kann dann mittels der SDO-Syntax für PHP gearbeitet werden. Beispielsweise wird in Zeile 7 der Nachname des Kunden auf `Mustermann` gesetzt.

Migration existierender PHP-Webanwendungen

Vorbemerkung Die folgenden Ausführungen geben lediglich einen groben Überblick über die mögliche Vorgehensweise zur Migration existierenden PHP-Webanwendungen. Sie sind als Denkanstoß und nicht als konkrete Anleitung zu verstehen. Die folgenden Ausführungen setzen darüber hinaus voraus, dass die zu migrierende PHP-Webanwendung eine saubere Trennung zwischen Anzeige (**'view'**) und Geschäftslogik (**'controller'**) einhält. Im Folgenden wird lediglich auf die Migration der Geschäftslogik eingegangen, die Anzeigelogik kann bei sauberer Trennung der Komponenten durch Entwicklung eines entsprechenden Adapters erhalten bleiben.

Durch die Integration von PHP in BPEL und in den WPS ergibt sich eine interessante Möglichkeit existierende PHP-Webanwendungen schrittweise auf BPEL-Geschäftsprozesse zu migrieren. Die Vorgehensweise ist dabei recht simpel: In einem ersten Schritt wird die gesamte Logik, die zur Abbildung eines Geschäftsprozesses nötig ist, auf ein oder mehrere PHP-Snippets im BPEL-Geschäftsprozesses übertragen.
Aus Gründen der Einfachheit wird an dieser Stelle angenommen, dass sich die dafür nötige Logik in einem einzigen PHP-Skript befindet. Die Migration der Geschäftslogik kann dann wie in Abbildung 7.2 dargestellt ist, vonstatten gehen:

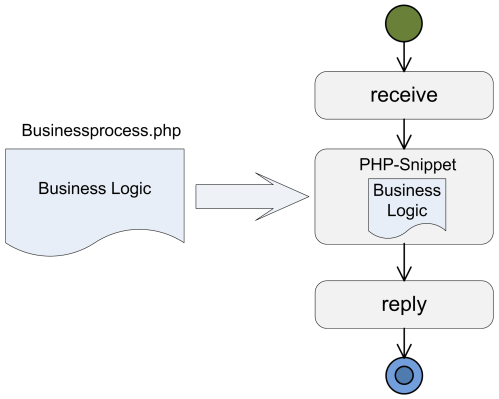

Abbildung 7.2: Migration einer PHP-Webanwendung auf BPEL

Sämtliche Geschäftslogik wird aus dem existierenden PHP-Skript so, wie sie ist, in einen BPEL-Geschäftsprozess übernommen. Dieser BPEL-Prozess enthält als einziges Element ein PHP-Snippet, welches die gesamte Geschäftslogik enthält.
Der Vorteil, der in der Folge realisiert werden kann liegt in den folgenden Punkten:

- Die Migration kann schrittweise verfeinert werden, sprich die existierende Geschäftslogik wird nach und nach auf mehrere PHP-Snippets verteilt, wobei Teile der Geschäftslogik in BPEL realisiert werden.
- Schleifen und Bedingungen können ab sofort über BPEL-Konstrukte realisiert werden. Auf diese Weise wird die Geschäftslogik schrittweise visualisiert.

- Zusätzlich benötigte Arbeitsschritte können in Form von normalen Service-Aufrufen über BPEL vorgenommen werden.

Statt des schrittweisen Vorgehens können diese Punkte selbstverständlich auch direkt bei der Migration berücksichtigt werden.

Durch die Migration werden folgende positive Eigenschaften der BPEL-Engine für den Geschäftsprozess verfügbar:

- Unterstützung für langlaufende Prozesse[1].
- Fehlerbehandlung inklusive Rollback und Compensation.
- Unterstützung für parallele Pfade.

Diese Eigenschaften lassen sich auch durch entsprechende PHP-Programme realisieren, jedoch bekommt man sie bei Verwendung des WPS ohne jeden Zusatzaufwand. Für die Realisierung paralleler Pfade können die entsprechenden Teile der Geschäftslogik auf parallel ablaufende PHP-Snippets verteilt werden, wie in Abbildung 7.3 angedeutet wird:

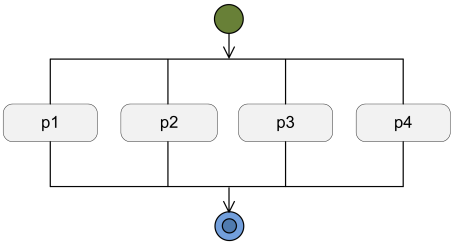

Abbildung 7.3: Verwendung paralleler Pfade

[1] Geschäftsprozesse laufen mitunter über sehr lange Zeitspannen. Beispielsweise läuft eine KFZ-Versicherung über Jahre hinweg. Erst durch das Eintreffen einer Schadensmeldung wird diese Prozessinstanz wieder aktiv

Abkürzungsverzeichnis

BPEL	Business Process Execution Language
BPC	Business Process Choreographer
DLL	Dynamic Link Library
EAR	Enterprise Application Archive
EMF	Eclipse Modelling Framework
JAR	Java Application Archive
JNI	Java Native Interface
JRE	Java Runtime Environment
PEAR	PHP Extension und Application Repository
PECL	PHP Extension Community Library
SDO	Service Data Objects
SOA	Service Oriented Architecture
WID	WebSphere Integration Developer
WPS	WebSphere Process Server
WSDL	Web Services Description Language

Listings

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[AAA+06] ALVES, Alexandre ; ARKIN, Assaf ; ASKARY, Sid ; BARRETO, Charlton ; BLOCH, Ben ; CURBERA, Francisco ; FORD, Mark ; GOLAND, Yaron ; GUÍZAR, Alejandro ; KARTHA, Neelakantan ; LIU, Canyang K. ; KHALAF, Rania ; KÖNIG, Dieter ; MARIN, Mike ; MEHTA, Vinkesh ; THATTE, Satish ; VAN DER RIJN, Danny Prasad Y. ; YIU, Alex. *Web Services Business Process Execution Language Version 2.0*. Committee Draft – WWW Resource http://www.oasis-open.org/committees/download.php/22036/wsbpel-specification-draft%20candidate%20CD%20Jan%2025%2007.pdf (zuletzt besucht 28.04.2007). 2006

[ABCG04] AUSTIN, Daniel ; BARBIR, Abbie ; CHRISTOPHER, Ferris ; GARG, Sharad. *Web Services Architecture Requirements*. W3C Working Group Note http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211 (zuletzt besucht 22.02.07). 2004

[ACD+03] ANDREWS, Tony ; CURBERA, Francisco ; DHOLAKIA, Hitesh ; GOLAND, Yaron ; KLEIN, Johannes ; LEYMANN, Frank ; LIU, Kevin ; ROLLER, Dieter ; SMITH, Doug ; THATTE, Satish ; TRICKOVIC, Ivana ; WEERAWARANA, Sanjiva. *Business Process Execution Language for Web Services Version 1.1*. IBM Developerworks, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf (zuletzt besucht 21.03.2007). 2003

[Aul05] AULKE, Gaylord: *Comparing PHP, Java, ASP*. 100 days Software, 2005. – WWW Resource http://www.100days.de/download.php?id=i/677.pdf (zuletzt besucht 28.04.2007)

[Bal96] BALZERT, Helmut: *Lehrbuch der Software-Technik*. 1.Auflage. Spektrum Akademischer Verlag, 1996. – ISBN 3–8274–0042–2

[BBC06] BLANVALET, Stany ; BOLIE, Jeremy ; CARDELLA, Michael: *BPEL Cookbook: Best Practices for SOA-Based Integration and Composite Applications Development (Taschenbuch)*. 1.ed. Packt Publishing, 2006. – ISBN 1904811337

[BBNP03] BATTY, John ; BRODSKY, Stephen ; NALLY, Martin ; PATEL, Rahul: *Next-Generation Data Programming: Service Data Objects*. BEA Systems Inc. and International Business Machines Corporation, 2003. – WWW Resource http://ftpna2.bea.com/pub/downloads/commonj/Next-Gen-Data-Programming-Whitepaper.pdf (zuletzt besucht 13.04.2007)

[BK06] BÖEKEMEIER, Jost ; KOERBER, Jon. *php/Java Bridge*. WWW Resource http:

//php-java-bridge.sourceforge.net/pjb/index.php (zuletzt besucht 14.04.2007). 2006

[BKV05] BISCHOFF, Georg ; KERSTEN, Roland ; VETTER, Thorsten. *Vergleich von BPEL-Workflow Modellierungstools*. Fachstudie – Universität Stuttgart. 2005

[Blo05] BLOOMBERG, Jason. *Service-Oriented Architecture: Why and How?* ZapThink Whitepaper. 2005

[Bol01] BOLTON, Fintan: *Pure Cobra (Pure Series)*. 1.ed. Sams, 2001. – ISBN 0672318121

[BPSM+06] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C.M ; MALER, Eve ; YERGEAU, François. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation – WWW Resource http://www.w3.org/TR/2006/REC-xml-20060816/ (zuletzt besucht 28.04.2007). 2006

[BS03] BATTY, Christopher ; SCUSE, David: *Using the Java Native Interface*. Department of Computer Science, University of Manitoba, 2003. – WWW Resource http://www.cs.umanitoba.ca/~eclipse/8-JNI.pdf (zuletzt besucht 28.04.2007)

[Bur04] BURGHARDT, Markus: *Web Services*. 1.Ausgabe. Deutscher Universitäts-Verlag, 2004. – ISBN 3–8244–2189–5

[CGV+02] CHESSELL, M. ; GRIFFIN, C. ; VINES, D. ; BUTLER, M. ; HENDERSON, C. Ferreira P. *Extending the concept of transaction compensation*. IBM SYSTEMS JOURNAL, VOL 41, NO 4. 2002

[DJMZ05] DOSTAL, Wolfgang ; JECKLE, Mario ; MELZER, Ingo ; ZENGLER, Barbara: *Service-orientierte Architekturen mit Web Services*. 1.Ausgabe. Spektrum Akademischer Verlag, 2005. – ISBN 3–84274–1457–1

[EE98] EDDON, Guy ; EDDON, Henry: *Inside Distributed COM*. 1.ed. Microsoft Press Corp., 1998. – ISBN 1572318496

[Emm00] EMMERICH, Wolfgang: *Engineering Distributed Objects*. 1.ed. John Wiley & Sons, 2000. – ISBN 0471986577

[GHM+03] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAUM, Jean-Jacques ; NIELSEN, Henrik F. *SOAP Version 1.2 Part 1: Messaging Framework*. WWW Resource http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ (zuletzt besucht 21.03.2007). 2003

[GMR+06] GUCKER, Joachim ; MÜLLER, Michael ; RAGER, Dietmar ; SCHÄFFER, Steffan ; SCHILDER, Walter ; THURNE, Veronika ; WINKLER, Dina: *Webanwendungen mit IBM Rational und IBM WebSphere V6*. Addison-Wesley, 2006. – ISBN 3–8273–2230–8

[Gol06] GOLEMON, Sara: *Extending and Embedding PHP*. 1.ed. Sams Publishing, 2006. – ISBN 0–672–32704

[Hab06] HABERKERN, Timo. *Gegensätze ziehen sich an – Java-Bibliotheken und -Anwendungen aus PHP heraus nutzen*. WWW Ressource http://phpmagazin.de/itr/online_artikel/psecom,id,845,nodeid,62.html (zuletzt besucht 16.04.2007). 2006

[Hal06] HALSBAND, David. *Wiederverwendbarkeit in Service-orientierten Architekturen*. Diplomarbeit – Universität Tübingen. 2006

[HRS06] HANTSCHEL, Ralf ; RUF, Fabian ; STROTBEK, Haiko. *Vergleich von BPEL Laufzeitumgebungen*. Fachstudie – Universität Stuttgart. 2006

[IBM06a] *IBM WebSphere Integration Developer documentation*. WWW Resource http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.6012.nav.doc/topics/welcome.html (zuletzt besucht 26.04.2007). 2006

[IBM06b] *IBM WebSphere Process Server documentation*. WWW Resource http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.6012.nav.doc/topics/welcome.html (zuletzt besucht 26.04.2007). 2006

[IBM07] *Business Process Choreographer Samples*. WWW Ressource http://publib.boulder.ibm.com/bpcsamp/index.html (zuletzt besucht 21.03.2007). 2007

[Kan06] KANG, Seung-Hwan. *PHP5 & Java integration on Win32 and examples*. WWW Resource http://www.dsl.uow.edu.au/~sk33/phpjava.htm (zuletzt besucht 14.04.2007). 2006

[KBS04] KRAFZIG, Dirk ; BANKE, Karl ; SLAMA, Dirk: *Enterprise SOA. Service Oriented Architecture Best Practices*. 1.ed. Prentice Hall International, 2004. – ISBN 0131465759

[KKL+05] KLOPPMANN, Matthias ; KOENIG, Dieter ; LEYMANN, Frank ; PFAU, Gerhard ; RICKAYZEN, Alan ; VON RIEGEN, Claus ; SCHMIDT, Patrick ; TRICKOVIC, Ivana: *WS-BPEL Extension for People - BPEL4People*. 2005. – WWW Resource ftp://www6.software.ibm.com/software/developer/library/ws-bpel4people.pdf (zuletzt besucht 28.04.2007)

[Lia99] LIANG, Sheng: *The Java Native Interface - Programmers's Guide and Specification*. Addison-Wesley, 1999. – WWW Resource http://java.sun.com/docs/books/jni/download/jni.pdf (zuletzt besucht 28.04.2007). – ISBN 0–201–32577–2

[Mar05] MARKS, Michael. *Modellierung und Automatisierung von Web Service Resourcen am Beispiel von BPEL Prozessen*. Diplomarbeit – Universität Stuttgart. 2005

[McC07] MCCABE, Joe: *Using PHP on Sun Java System Web Server*. Sun Developer Network, 2007. – WWW Resource http://developers.sun.com/webserver/reference/techart/php2.html (zuletzt besucht 28.04.2007)

[PB04] PORTIER, Bertrand ; BURDINSKY, Frank: *Introduction to Service Data Objects*. International Business Machines Corporation, 2004. – WWW Resource http://www-128.ibm.com/developerworks/java/library/j-sdo/ (zuletzt besucht 28.04.2007)

[RD00] RADIYA, Ashvin ; DIXIT, Vibha. *The basics of using XML Schema to define elements*. IBM Developer Works – WWW Resource http://www-128.

ibm.com/developerworks/xml/library/xml-schema/ (zuletzt besucht 28.04.2007). 2000

[Sch04] SCHLOSSNAGLE, George: *Advanced PHP Programming*. Sams Publishing, 2004. – ISBN 0–672–32561–6

[Sto04] STOGOV, Dmitry: *PHP SOAP Extension*. Zend Technologies Ltd, 2004. – WWW Resource http://devzone.zend.com/node/view/id/689 (zuletzt besucht 28.04.2007)

[WCL+05] WEERAWARANA, Sanjiva ; CURBERA, Fransisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture*. 1.Ausgabe. Prentice Hall PTR, 2005. – ISBN 0–13–148874–0

[WLP+06] WAHLI, Ueli ; LEYBOVICH, Larissa ; PREVOST, Eric ; SCHER, Russel ; ANDRE, Venancio ; WIEDERKOM, Sascha ; MACKINNON, Neil: *Business Process Management: Modeling through Monitoring Using WebSphere V6 Products*. 1.ed. IBM Redbooks, 2006. – ISBN 0738496472

[Wol05] WOLFF, Markus. *Applikationsintegration mit PHP 5*. WWW Ressource http://opensource.21st.de/static/phpintegration.pdf (zuletzt besucht 16.04.2007). 2005

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig und nur mit Hilfe der angegebenen Quellen und Hilfsmittel verfasst zu haben. In dieser Arbeit werden Warenzeichen ohne besondere Kennzeichnung verwendet. Diese Warenzeichen unterliegen geltendem Recht.

Tübingen, 23. Mai 2007 Volker Klaeren