

Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3227

**Aufruf und visuelle Korrelation
von wissenschaftlichen Workflows
in einem Workflow
Modellierungswerkzeug**

Aleksandar Tolev

Studiengang:	Softwaretechnik
Prüfer:	Jun.-Prof. Dr.-Ing Dimka Karastoyanova
Betreuer:	Dipl.-Inf. Mirko Sonntag

begonnen am:	02. Mai 2011
beendet am:	01. November 2011

CR-Klassifikation:	H.4.1, H.5.2, K.1
---------------------------	-------------------

Inhaltsverzeichnis

1. Einleitung	7
1.1. Aufgabenstellung	8
1.2. Gliederung	8
2. Grundlagen	9
2.1. Workflows	9
2.1.1. Definition	9
2.1.2. Workflowkategorien	11
2.1.3. Lebenszyklus	14
2.2. Scientific Workflows	15
2.2.1. Scientific Lebenszyklus	16
2.2.2. Unterschiede und Gemeinsamkeiten	18
2.3. Service Oriented Architecture	19
2.3.1. Definition	19
2.3.2. Web Service	21
2.4. Business Process Execution Language	22
2.4.1. Definition	23
2.4.2. Infrastruktur	23
2.4.3. BPEL Modell	24
2.5. Verwendete Werkzeuge	26
2.5.1. Eclipse BPEL Designer	26
2.5.2. Apache ODE	28
2.5.3. BPEL Event Modell	30
3. Verwandte Arbeiten	33
3.1. Entwicklung eines Monitoring-Tools zur Unterstützung von parametrisierten Web Service Flows	33
3.2. Interaktives Monitoring von wissenschaftlichen Workflows	33
3.3. Ein Event Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE	33
3.4. A Monitoring Tool for the Apache ODE BPEL Engine	34
3.5. Integration of the Result Visualization into a Workflow Modeling Tool	34
3.6. Sedna	34
3.7. Microsoft Trident	35
4. Anforderungen	37
4.1. Rahmenbedingungen	37
4.2. Anforderungen	37

5. Konzept	39
5.1. Parametrisiertes Starten von Workflows	39
5.1.1. Ist-Zustand	39
5.1.2. Konzept für die Parametrisierung	40
5.2. Visuelle Korrelation von parallelen Workflows	46
5.2.1. Architektur	47
5.2.2. Monitor Manager und Monitoring Provider	48
5.2.3. Weitere Manager	48
5.2.4. XPathMapper und XPathMap	49
5.2.5. Auditing	49
6. Implementierung	53
6.1. Parametrisiertes Starten von Workflows	53
6.1.1. Parser	53
6.1.2. Parameterdialog	56
6.1.3. Parameter Handler	58
6.1.4. Nachrichtenaufbau mit Hilfe des kartesischen Produkts	60
6.1.5. Apache ODE	61
6.2. Visuelle Korrelation von parallelen Workflows	62
6.2.1. Monitoring Provider	63
6.2.2. Monitor Manager	64
6.2.3. Instance Manager	65
6.2.4. XPathMapper und XPathMap	66
6.2.5. Auditing	67
6.3. Anwendungsbeispiel	70
7. Zusammenfassung und Ausblick	77
7.1. Ausblick	77
A. Anhang	79
A.1. WSDL	79
A.2. SOAP Nachrichten	80
A.3. BPEL	82
Literaturverzeichnis	85

Abbildungsverzeichnis

2.1. Beispielprozess	10
2.2. Bewerbungsprozess	10
2.3. Geschäftsprozess und Workflows	12
2.4. Klassifizierung der Workflows	13
2.5. Workflow Lebenszyklus nach Sonntag	14
2.6. Business Process Lifecycle nach Weske	15
2.7. Scientific Workflow Lebenszyklus nach Ludäscher	17
2.8. Scientific Workflow Lebenszyklus nach Sonntag	18
2.9. SOA Dreieck	20
2.10. Web Service Architektur	22
2.11. BPEL Infrastruktur	23
2.12. Apache ODE Architektur	29
2.13. Zustandsdiagramm eines Prozesses	31
5.1. Starten eines Workflows	40
5.2. Parametrisierter Aufruf eines Workflows	41
5.3. Darstellung des Ablaufs für parametrisiertes Starten	42
5.4. Normaler Ablauf einer Receive/Pick Aktivität	45
5.5. Neuer Ablauf einer Receive/Pick Aktivität	46
5.6. Architektur des Monitoringtools	47
5.7. Architektur des neuen Konzepts	51
6.1. Erweiterung des BPEL Schemas	54
6.2. Check Box zur Markierung von Parametervariablen	55
6.3. Parameterdialog	58
6.4. Klassendiagramm des parametrisierten Aufrufs	61
6.5. Klassendiagramm Monitor Manager, Monitor Provider und Instance Information	65
6.6. Datenbanktabellen des Auditing Tool	67
6.7. Prozess zur Visualisierung von Luftströmen	71
6.8. Markierung der Variablen Variablen zu Parametervariablen	72
6.9. Parameterdialog zur Eingabe der Parameterwerte	73
6.10. Paralleles Überwachen von Workflows	75

Verzeichnis der Listings

5.1. Schema der Nachricht, die an die Workflow-Engine gesendet wird	43
6.1. Setzen des parameter Attributs	54
6.2. Setzen der Check Box und Änderung des Variable Modells	56
6.3. Parsen des Prozesses nach Parametervariablen und Aufbau der Komponenten im Dialog	57
6.4. Iteration über die Liste und Aufruf der buildEntry() Methode	57
6.5. Hinzufügen der jeweiligen Komponenten in die Entry Oberklasse	59
6.6. Initialisierung des Parameterdialogs	59
6.7. Kartesisches Produkt	60
6.8. Befüllen der Parametervariablen	62
6.9. Extraktion und Speicherung des korrekten Knotens	63
6.10. Bauen eines DOM-Objekts aus dem extrahierten Knoten	63
6.11. Finden einer Instanz anhand ihrer ID	64
6.12. Bearbeitung eines undeployten Prozesses	66
6.13. Extraktion der notwendigen Daten für die Auditing Sicht	68
6.14. Anpassung der Sicht im Eclipse BPEL Designer	69
6.15. Erste Nachricht, die an die Apache ODE gesendet wird	72
6.16. Zweite Nachricht, die an die Apache ODE gesendet wird	73
A.1. Erste SOAP Nachricht aus dem kartesischem Produkt	81
A.2. Zweite SOAP Nachricht aus dem kartesischem Produkt	81

1. Einleitung

Beim Begriff Geschäftsprozess assoziiert man meistens einen typischen Prozess innerhalb einer Bank oder mit einer Versicherung, wie z.B. das Prüfen der Kreditwürdigkeit eines Klienten bei der Bank oder die Schadensersatzforderung bei der Versicherung. Häufig ist das Ergebnis ein Stapel voller Papiere. Diese Aktivitäten werden nach dem gleichen Muster wiederholt ausgeführt, das sogenannte Prozess Modell. Geschäftsprozesse beziehen sich dabei auf die wirkliche Welt und spiegeln Arbeitsschritte wider. Oft werden die Prozesse ohne Hilfe von Computern abgewickelt, da es nicht notwendig ist, einen Computer einzusetzen. Wenn doch mal Prozesse auf dem Computer durchgeführt werden müssen, dann spricht man nicht mehr von Geschäftsprozessen, sondern von Workflow Modellen [LRoo].

In den letzten Jahren wurde der Workflow Technologie mehr und mehr Beachtung im wissenschaftlichen Bereich geschenkt. Daraus resultierte der Begriff **scientific workflows**. So können durch wissenschaftliche Workflows, komplexe Kalkulationen durchgeführt werden, wie z.B. die Berechnung von partiellen Differentialgleichungen, die für die Bestimmung zeitlicher und räumlicher Änderungen von simulierten Objekten notwendig sind [GSK⁺11].

Da sich die traditionelle Workflow Technologie im wirtschaftlichen Bereich etabliert hat, liegt es nahe, diese Technologie ebenfalls für den wissenschaftlichen Bereich zu verwenden. Der Einsatz der Workflow Technologie soll Wissenschaftlern die Möglichkeit geben, sich mehr auf ihre Arbeit zu konzentrieren, während viele Arbeitsschritte im Laufe der Entwurfs- und Ausführungsphase automatisiert werden [GSK⁺11].

So wie es Gemeinsamkeiten zwischen der normalen Workflow Technologie und den wissenschaftlichen Workflows gibt, so gibt es ebenfalls Unterschiede, welche die Anwendung der traditionellen Technologie im wissenschaftlichen Bereich erschwert. Die wissenschaftlichen Workflows besitzen andere Anforderungen an die Technologie, als die traditionellen Workflows, sodass die Technologie zunächst an die Anforderung der wissenschaftlichen Workflows angepasst werden muss, bevor man sie verwenden kann. So besitzen Wissenschaftler weniger technisches Wissen und haben eine andere Herangehensweise an Probleme als Business Spezialisten, sodass sie in diesem Bereich mehr Unterstützung benötigen [LWMB09]. Eine genau Beschreibung der unterschiedlichen Anforderungen befindet sich in Kapitel 2.

Die Anforderungen der wissenschaftlichen Workflows müssen analysiert und die traditionelle Workflow Technologie adaptiert werden, um wissenschaftliche Workflows zu beschreiben und auszuführen.

Das Institut für Architektur von Anwendungssystemen, an der Universität Stuttgart, arbeitet im Rahmen von SimTech¹ an einer Workflow-Umgebung für wissenschaftliche Workflows, basierend auf der traditionellen Workflow Technologie [GSK⁺11].

1.1. Aufgabenstellung

Ziel dieser Diplomarbeit ist es, ein Konzept für das Parametrisieren von wissenschaftlichen Workflows zu entwickeln. Beim Starten der Workflows, aus dem Modellierungswerkzeug heraus, soll der Wissenschaftler nach den Parametern abgefragt werden. Es soll die Möglichkeiten geboten werden, Parameterstudien zu starten. Je nach Spezifizierung der Parameter werden daraufhin ein oder mehrere Workflow Instanzen gestartet. Des weiteren soll ein Konzept gefunden werden, das Wissenschaftlern ermöglicht, gleichzeitig an mehreren unterschiedlichen Workflow Modellen und Workflow Instanzen zu arbeiten. Dazu wird ein Korrelationsmechanismus benötigt, um die Vorgänge in der Workflow-Engine den Workflows Modellen und den zugehörigen Instanzdaten im Modellierungswerkzeug korrekt zuzuordnen. Beide Konzepte sollen für den Eclipse BPEL Designer, als Modellierungswerkzeug, und die Apache ODE, als Workflow-Engine, implementiert werden.

1.2. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden die Grundlagen dieser Arbeit beschrieben

Kapitel 3 – Verwandte Arbeiten: Führt verwandte Arbeiten auf, die sich mit Teilaspekten dieser Arbeit beschäftigen

Kapitel 4 – Anforderungen: Hier werden die Anforderungen, die an die Konzepte gestellt werden, beschrieben

Kapitel 5 – Konzept: In diesem Kapitel werden die beiden Konzepte zur Lösung des Problems vorgestellt und erläutert

Kapitel 6 – Implementierung: Hier wird die Realisierung der im Kapitel 5 vorgestellten Konzepte beschrieben

Kapitel 7 – Zusammenfassung und Ausblick: Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor

¹<http://www.simtech.uni-stuttgart.de/>

2. Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt, die zum Verständnis dieser Arbeit beitragen. Es wird auf die für diese Arbeit wichtigen Aspekte eingegangen. Weitere Informationen sind in der referenzierten Literatur zu finden.

2.1. Workflows

Im folgenden Abschnitt werden Workflows definiert und beschrieben. Es wird auf ihren Lebenszyklus eingegangen und auf ihre Kategorisierung.

2.1.1. Definition

Wie in der Einleitung bereits erwähnt, assoziiert man mit Geschäftsprozessen z.B. die Abwicklung eines Kredites bei der Bank oder die Schadensersatzforderung bei der Versicherung. Der Prozess wird in einzelne Teilschritte oder Aktivitäten unterteilt. Die Teilschritte sind miteinander verbunden und bilden am Ende ein Produkt. Laut [Wes07] wird ein **business process** wie folgt definiert:

A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.

In Abbildung 2.1 wird ein Geschäftsprozess dargestellt. Somit enthält ein Geschäftsprozess eine Menge an Aktivitäten, die innerhalb einer organisierten und technischen Umgebung in Verbindung stehen. Ebenfalls können Geschäftsprozesse von einer einzelnen Organisation erstellt werden, aber auch mit Prozessen anderer Organisationen interagieren.

In Abbildung 2.2 wird ein Prozess dargestellt, der mit einem anderen Prozess einer anderen Organisation interagiert.

Solche Prozesse können mehrfach wiederholt werden. Dafür ist eine Art Schablone notwendig, welche den Prozess allgemein beschreibt [LR00]. Eine Schablone ist das Prozessmodell, das Weske wie folgt definiert [Wes07]:

A business process model consists of a set of activity models and execution constraints between them.

2. Grundlagen

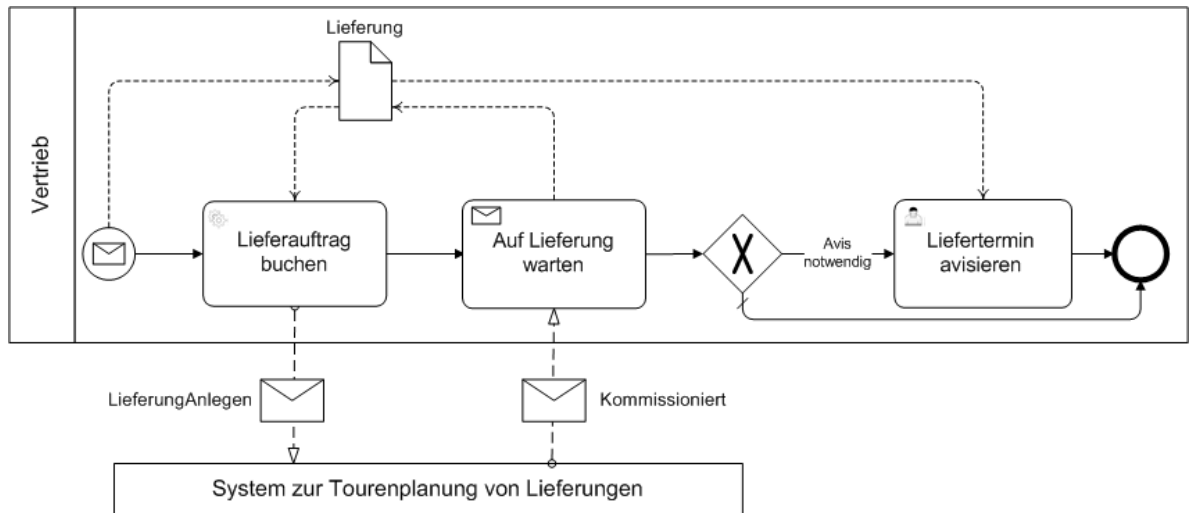


Abbildung 2.1.: Beispielprozess^a

^a<http://www.bpm-guide.de/wp-content/uploads/2009/08/bpmn20beispiel.png>

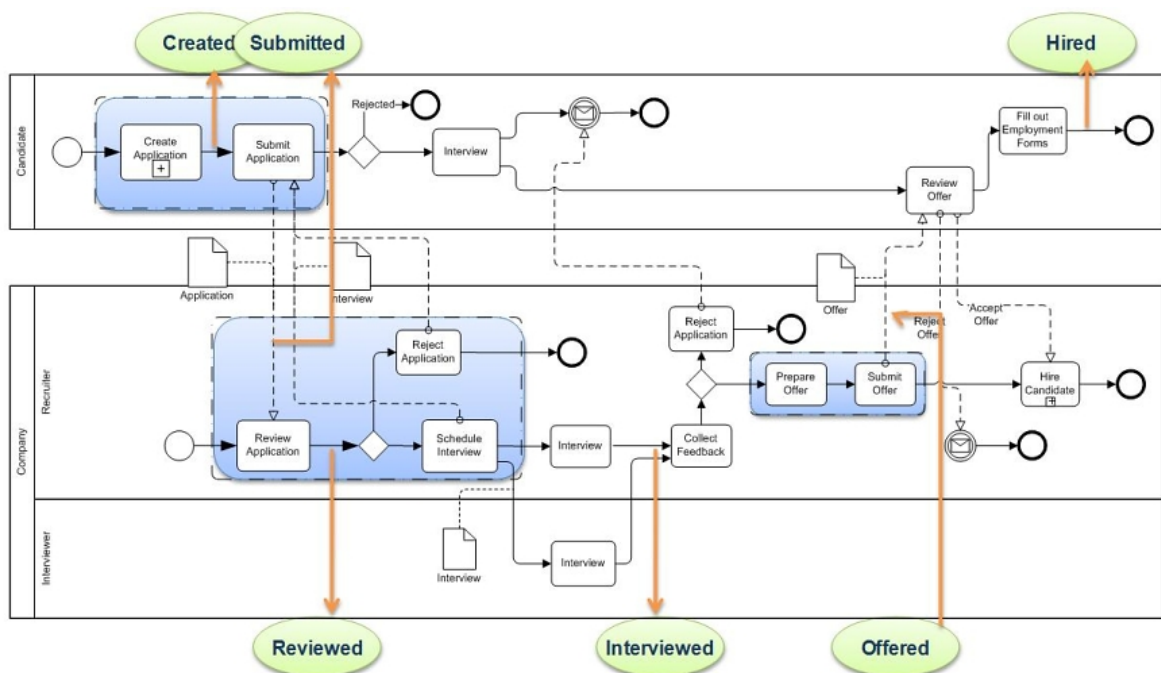


Abbildung 2.2.: Bewerbungsprozess^a

^ahttp://www.infoq.com/resource/articles/seven-fallacies-of-bpm/en/resources/job_application_process.jpg

Die oben erwähnte Definition ähnelt der Definition eines Geschäftsprozesses, allerdings mit dem Unterschied, dass im Prozessmodell von einer Menge von Aktivitätsmodellen die Rede ist und dass zusätzlich Bedingungen zur Ausführung zwischen den Aktivitätsmodellen existieren. Ein Aktivitätsmodell ist ebenfalls, wie ein Prozessmodell, nur eine Schablone, die zwar beschreibt, was die Aktivität kann, aber noch keine Ausführung der Aktivität darstellt.

Die Ausführung eines Prozessmodells wird als Prozessinstanz bezeichnet. Eine Prozessinstanz ist ein konkreter Ausführungsfall eines Prozessmodells. Eine Prozessinstanz enthält ebenfalls Instanzen der Aktivitäten, die durch das Aktivitätenmodell im Prozessmodell vorgegeben sind. Somit beschreibt das Prozessmodell abstrakt einen Geschäftsprozess, während die Instanz die konkrete Beschreibung eines Geschäftsprozesses ist. Das Gleiche gilt für die Aktivitäten [Wes07].

Somit beschreibt das Prozessmodell die Struktur eines Geschäftsprozesses in der realen Welt (Abwicklung des Kredites, Schadensersatzforderung, Bewerbungsprozess). Solche Prozesse müssen nicht unbedingt mit Hilfe eines Computers durchgeführt werden. Viele Prozesse werden manuell durchgeführt. Ein Beispiel hierfür geben Leymann und Roller [LR00]: Das Verteilen eines Dokumentes an die Mitarbeiter durch den Manager. Erst der Manager bestätigt, ob ein Dokument weiter gereicht wird oder nicht. Somit entscheidet der Manager und nicht der Computer.

Allerdings gibt es Prozesse, bei dem ein Teil des Prozesses oder der gesamte Prozess durch Computer bearbeitet wird. Andere Teile wiederum, werden manuell durchgeführt. Die Teile eines Prozesses, die mit Hilfe eines Computers erledigt werden, nennt man Workflow Modell, wie die Abbildung 2.3 zeigt [LR00].

Weske liefert für den Begriff Workflow noch eine konkrete Definition [Wes07].

Workflow is the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

Somit definieren Weske, Leymann und Roller einen Workflow als ein (Teil-)Prozess, der durch den Computer automatisiert bearbeitet wird.

Das Prinzip des Prozessmodells ist auf Workflows übertragbar. In dieser Arbeit werden die Begriffe Geschäftsprozess, Prozess, business process und Workflow als Synonyme verwendet. Falls es einer eindeutigen Bezeichnung bedarf, wird der jeweilige Begriff verwendet.

2.1.2. Workflowkategorien

Workflows können in Kategorien aufgeteilt werden. Die GIGA Information Group unterteilt Workflows in vier Kategorien [LR00]. Die Workflows werden bezüglich ihres Wertes im Geschäftsbereich und ihrer Anwendungshäufigkeit unterteilt. Abbildung 2.4 zeigt diese Klassifizierung der Workflows.

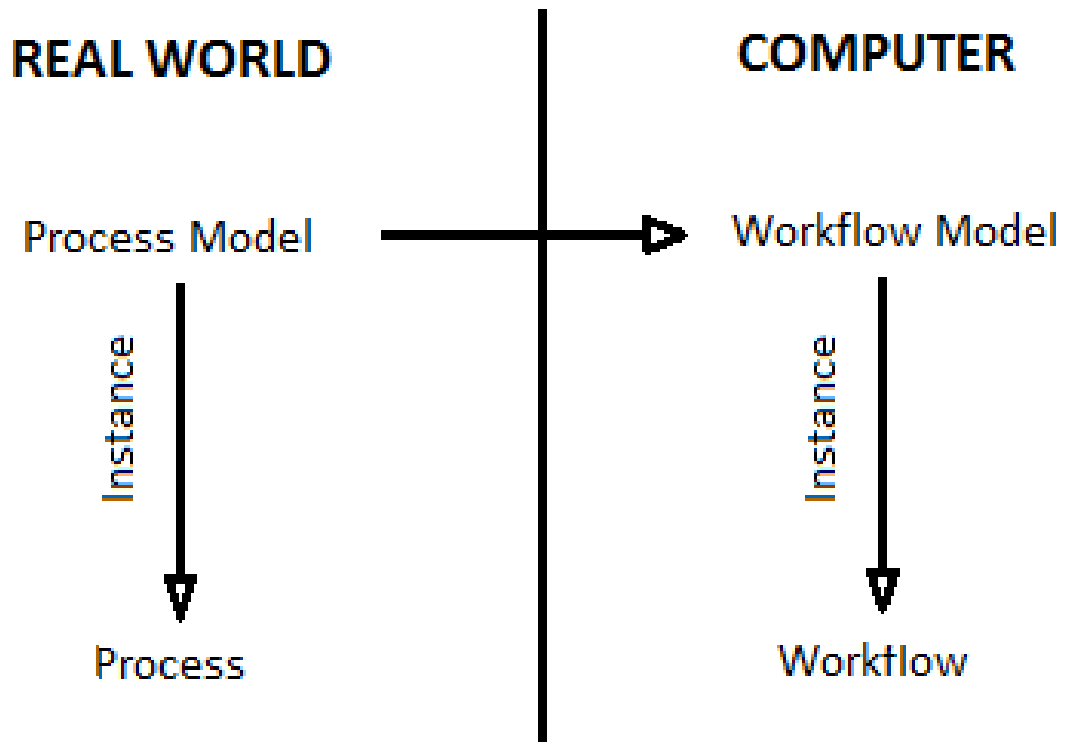


Abbildung 2.3.: Geschäftsprozess und Workflows [LRoo]

Die y-Achse beschreibt die Wichtigkeit (business value) eines Workflows innerhalb eines Unternehmens. Ein hoher business value beschreibt eine Kernkompetenz und ist essentiell für das Unternehmen. Durch diese Workflows wird die Unternehmenskultur definiert. Die x-Achse beschreibt die repetition. Sie gibt an, wie häufig ein bestimmter Workflow ausgeführt wird. Der Wert der repetition ist ein Indikator dafür, ob es sich lohnt einen Workflow zu modellieren oder nicht. Denn die Workflowmodellierung für einen noch nicht existierenden Workflow ist zeitaufwändig und kostspielig [LRoo].

Ausgehend von diesen beiden Charakteristika, können die Workflows in vier unterschiedliche Kategorien eingeteilt werden [LRoo].

- I Collaborative Workflows haben einen hohen Stellenwert innerhalb des Unternehmens, werden jedoch selten ausgeführt. Der Workflow ist sehr komplex und wird ausgehend von einem Projektplan, für eine bestimmte Aufgabe erstellt. Ein solcher Workflow ist z.B. das Erstellen des technischen Handbuchs für eine Software.

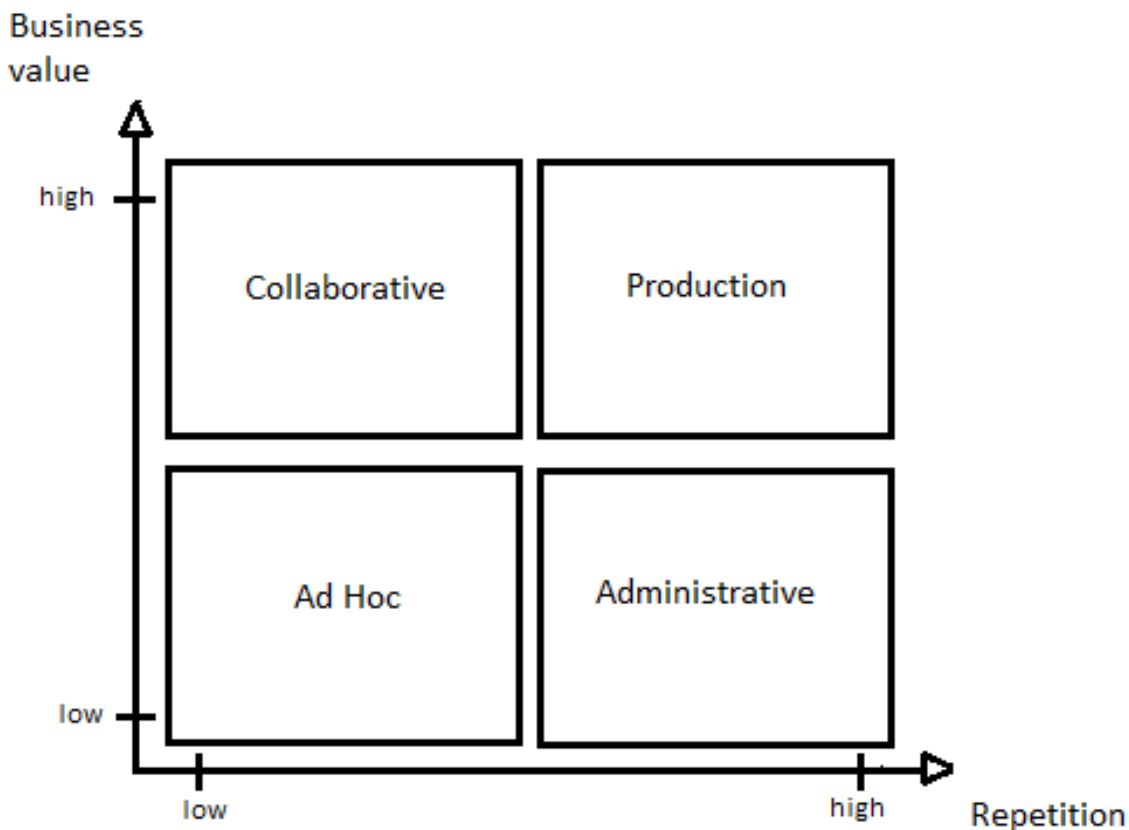


Abbildung 2.4.: Klassifizierung der Workflows [LRoo]

- II Ad Hoc Workflows haben einen geringen Stellenwert innerhalb des Unternehmens und werden selten ausgeführt. Sie werden dann von beteiligten Personen oder Prozessen ausgeführt, wenn sie benötigt werden. So fällt z.B. das Weiterleiten einer Nachricht unter diese Kategorie. Wenn der Bedarf vorhanden ist, wird die Nachricht weiter geleitet, ansonsten terminiert der Workflow an der Stelle, an der die Nachricht nicht weiter geleitet wird.
- III Administrative Workflows haben einen geringen Stellenwert innerhalb des Unternehmens, werden jedoch sehr häufig ausgeführt. Das sind die Workflows, die man im Allgemeinen kennt, z.B. der Bestellvorgang im Internet oder die Spesenabrechnung.
- IV Production Workflows haben einen hohen Stellenwert innerhalb des Unternehmens und werden häufig ausgeführt. Diese Workflows beschreiben die Kernkompetenz eines Unternehmens. Dazu gehören z.B. die Kreditabwicklung bei der Bank oder die Schadensersatzforderung bei der Versicherung. Production Workflows sind oft nicht richtig fest in einem Unternehmen integriert, da bestehende Workflows übernommen und nicht innerhalb einer geeigneten Umgebung integriert werden.

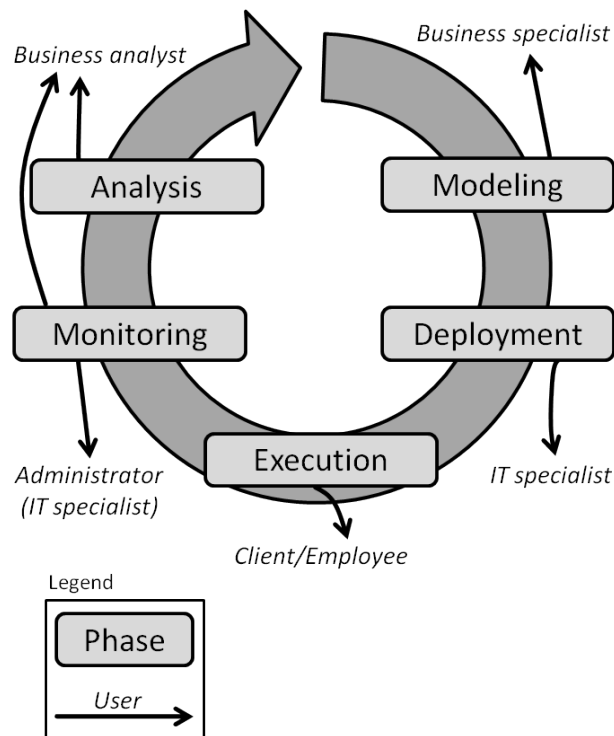


Abbildung 2.5.: Workflow Lebenszyklus nach Sonntag [SKL10]

2.1.3. Lebenszyklus

Jeder Workflow, der ausgeführt wird, durchläuft bestimmte Phasen. Diese Phasen sind im **Business Process Lifecycle** zusammengefasst. In Abbildung 2.5 sind die fünf Phasen des Lebenszyklus dargestellt.

Der Lebenszyklus eines Workflows besteht aus fünf Phasen. Beginnend mit der Modellierung des Workflows, durch einen Spezialisten mit IT Erfahrung, wird der neu modellierte Workflow auf eine Engine hochgeladen und deployt. Eine Engine ist ein Server, der die eingehenden Workflows ausführt und abarbeitet. Nach dem Hochladen des Workflows wird die Ausführung entweder durch eine Person oder durch eine eingehende Nachricht angestoßen. Der Workflow wird gestartet und durchläuft alle Aktivitäten, die innerhalb des Workflows beschrieben sind. Während der Ausführung werden Daten über die Workflow-Instanz gesammelt und angezeigt. Diese Daten können z.B. der Zustand einer Aktivität oder der Instanz sein, aber auch bestimmte Werte für bestimmte Variablen, die verwendet werden. Die Datensammlung und die Anzeige der Daten werden in der Phase Monitoring zusammengefasst. In der letzten Phase wird durch einen Analysten festgestellt, ob man Änderungen am Modell vornehmen muss und ob die Daten für den Geschäftsbereich relevant sind. Bei Änderungen am Modell kann der Workflow erneut gestartet werden und er durchläuft den kompletten Lebenszyklus ein weiteres Mal [SK10].

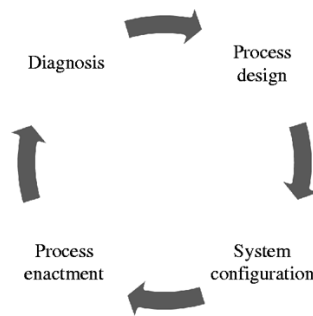


Abbildung 2.6.: Business Process Lifecycle nach Weske ^a

^ahttp://www.emeraldinsight.com/content_images/fig/1570150506002.png

Weske [Wes07] fasst diese fünf Phasen in vier zusammen. In Abbildung 2.6 wird der Lebenszyklus nach Weske dargestellt.

Die erste Phase, Entwurf (process design), entspricht der Phase Modeling. In dieser Phase werden die Anforderungen geklärt und daraus ein Geschäftsprozess modelliert. Neben der Modellierung findet auch die Validierung und Simulation des Prozesses statt. So wird sichergestellt, dass die Instanzen korrekt im Sinne der Prozessmodells sind.

Die zweite Phase, Konfiguration, beinhaltet das Deployment des modellierten Prozesses und bezieht sich auf die zweite Phase aus [SK10]. Hier wird das System konfiguriert und der Prozess hochgeladen.

Die dritte Phase, Enactment, fasst die beiden Phasen Execution und Monitoring zusammen. Während der Ausführung werden Daten gesammelt und visualisiert dargestellt.

Als letzte Phase beschreibt Weske die Diagnose (diagnosis), welche der Analysephase aus [SK10] gleichkommt. Hier werden die Daten ausgewertet und eventuelle Änderungen am Modell vorgenommen.

Als Referenz eines Workflow Lebenszyklus wird der Lebenszyklus von [SK10] in dieser Arbeit weiter verwendet. Wenn in dieser Arbeit von einem geschäftlichen Lebenszyklus gesprochen wird, bezieht er sich auf diesen.

2.2. Scientific Workflows

In den letzten Jahren wurde das Wissenschaftsfeld revolutioniert. Mit der Einführung von Supercomputern, die mehr Berechnungen pro Sekunde ausführen können, konnte man größere und komplexere Systeme simulieren, die auf der Berechnung von partiellen Differentialgleichungen beruhen. Dadurch war es möglich, solche Gleichungen schnell zu lösen. Bis zu diesem Zeitpunkt waren solche Berechnungen entweder nicht möglich oder mit einem sehr hohen Zeitaufwand verbunden [TDGS07].

Eine weitere Änderung war die Einführung von verteilten Netzwerken. Man versuchte Ressourcen verteilt auf mehreren Rechnern zu halten und Berechnungen ebenfalls verteilt durchzuführen. Dadurch stieg der Wunsch nach Konsistenz, Parallelität, Synchronisation und Datenverwaltung [TDGS07].

Nachdem die Berechnung von großen und komplexen Daten ein immer wichtigerer Punkt in der Wissenschaft wurde, hatte man nach einer Möglichkeit gesucht, eine Technologie zu (er)finden, die mit diesen Berechnungen zurecht kommt. Die Technologie existierte bereits und erfüllte viele Anforderungen, die die Wissenschaft stellte [BG07]. Es ist die oben beschriebene Workflow Technologie. So schenkte man in den letzten Jahren, nicht nur im Geschäftsbereich den Workflows mehr Beachtung, sondern auch im wissenschaftlichen Bereich [TDGS07]. Das führte dazu, dass man den Begriff **scientific workflow** einführte. Der Begriff beschreibt alle Workflows, die im wissenschaftlichen Bereich ausgeführt werden.

In dieser Arbeit werden **scientific workflows** und wissenschaftliche Workflows als Synonyme verwendet.

Die **scientific workflows** sollen die Wissenschaftler bei ihrer Arbeit unterstützen und ihnen in gewissen Punkten die Arbeit abnehmen [SKL10].

Da sich der Geschäftsbereich und die Wissenschaft in manchen Punkten unterscheiden, muss die Workflow Technologie erst an die wissenschaftlichen Anforderungen angepasst werden, damit auch die Wissenschaftler sie anwenden können [GSK⁺11]. Zunächst wird auf den Lebenszyklus eingegangen, welche Gemeinsamkeiten bzw. Unterschiede der wissenschaftliche Lebenszyklus zum gängigen Lebenszyklus besitzt.

2.2.1. Scientific Lebenszyklus

Wie in Abschnitt 2.1.3 schon beschrieben, folgen Workflows einem bestimmten Lebenszyklus, der sich ständig wiederholt. Da wissenschaftliche Workflows andere Anforderungen haben, ist es nicht möglich, den Lebenszyklus der gängigen Workflow Technologie einfach zu übernehmen. Es bedarf eines erweiterten Lebenszyklus, der den Anforderungen gerecht wird [GSK⁺11].

In der Literatur gibt es viele unterschiedliche Definitionen des wissenschaftlichen Workflow Lebenszyklus. So beschreiben Ludäscher et al. [LWMB09] einen Lebenszyklus mit fünf Phasen, der dem Lebenszyklus der gängigen Workflow Technologie sehr nahe kommt. Die Abbildung 2.7 stellt den Lebenszyklus nach [LWMB09] dar.

Die erste Phase beschreibt ein Experiment, welches ausgeführt werden soll. Aus diesem Experiment wird dann ein Workflow entworfen. Wissenschaftler wollen dabei versuchen, bereits bestehende Templates zu verwenden, die sie entweder komplett übernehmen oder verändern können. Neue, noch nicht bestehende Workflow Entwürfe oder Ergebnisse können sie über ein gemeinsam genutztes Repository öffentlich machen und zur Weiterverwendung zur Verfügung stellen.

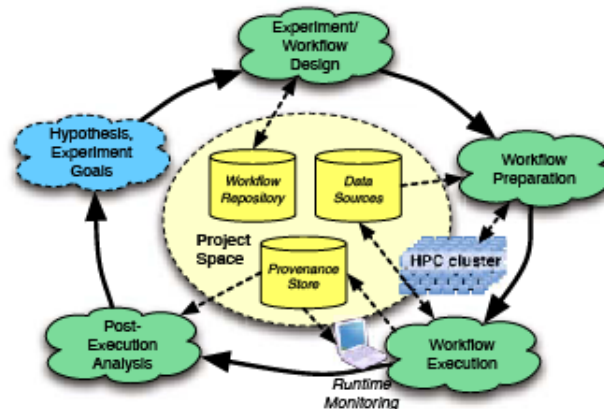


Abbildung 2.7.: Scientific Workflow Lebenszyklus nach Ludäscher [LWMB09]

Während der Workflow Preparation werden die Ressourcen für die Ausführung vorbereitet. Die Ressourcen werden meist aus verteilten Clustern bezogen. Hier gibt der Wissenschaftler auch seine Parameter ein, mit denen er seinen Workflow starten möchte.

Nach der Vorbereitung wird der Workflow ausgeführt. Hier werden die Daten aus der Vorbereitung verwendet und neue Ergebnisse geliefert. Bei langlebigen Prozessen ist ein Überwachen des Workflows essentiell. Daten zum Workflow werden angezeigt und der Wissenschaftler kann auf eventuelle Aktionen reagieren.

In der vorletzten Phase werden die Ergebnisse, die bei der Ausführung entstehen, analysiert. Hier wird analysiert, ob die Daten korrekt sein können, ob man bestimmte Aktivitäten aufgrund ihrer Performanz optimieren kann oder warum ein bestimmtes Experiment fehlgeschlagen ist. Ausgehend von dieser Analyse, können dann Hypothesen erstellt werden, die zu eventuellen Änderungen am Experiment und am Workflow führen [LWMB09].

Sonntag und Karastoyanova [SK10] beschreiben, basierend auf einem Beispiel für das Vorgehen eines Wissenschaftlers, ebenfalls einen Lebenszyklus. Da viele Wissenschaftler zuvor nicht wissen, was genau bei den Experimenten herauskommt, entwickeln sie ihre Workflows explorativ. Das bedeutet, dass sie ein Experiment mit gewissen Parametern starten und schauen welche Resultate dabei herauskommen. Danach werden neue Parameter verwendet usw. Dieses „praktische Herumprobieren“ wird auch als **trial-and-error** Ansatz bezeichnet. Der Lebenszyklus nach Sonntag und Karastoyanova ist in Abbildung 2.8 abgebildet.

Der Lebenszyklus besteht aus drei Phasen. Nachdem der Workflow modelliert ist, wird dieser ausgeführt. Während der Ausführung werden Daten über den Workflow gesammelt und angezeigt. Ein laufender wissenschaftlicher Workflow kann angehalten und fortgesetzt werden. Warum das der Fall ist wird in Abschnitt 2.2.2 genauer beschrieben. Nachdem der Workflow beendet ist, kommt die Analysephase. In dieser Phase werden die Ergebnisse analysiert und Schlüsse daraus gezogen [SK10].

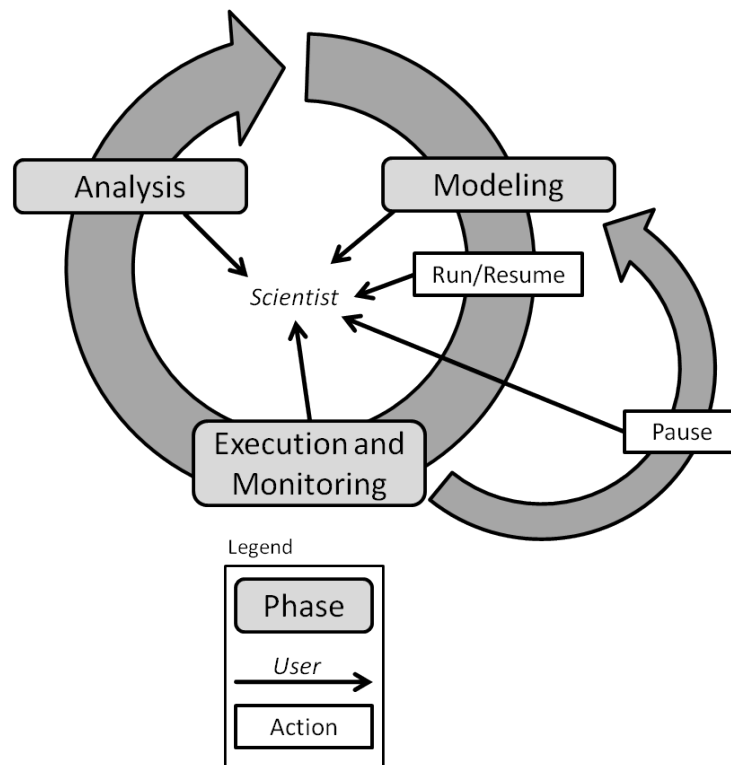


Abbildung 2.8.: Scientific Workflow Lebenszyklus nach [SKL10]

In dieser Arbeit wird die Definition eines wissenschaftlichen Workflow Lebenszyklus von [SK10] als Referenz verwendet. Wird von einem wissenschaftlichen Workflow Lebenszyklus gesprochen, bezieht er sich auf diesen.

2.2.2. Unterschiede und Gemeinsamkeiten

Nicht nur der Lebenszyklus unterscheidet sich von der gängigen Workflow Technologie. Es gibt bestimmte Anforderungen, die wissenschaftliche Workflows an die Technologie stellen. Allerdings gibt es auch Gemeinsamkeiten. In diesem Abschnitt werden die Unterschiede und Gemeinsamkeiten aufgelistet. Die Auflistung ist nicht vollständig. Sie soll nur exemplarisch Unterschiede und Gemeinsamkeiten zeigen.

Die gängige Workflow Technologie bietet schon eine Menge an Funktionalität, die für wissenschaftliche Workflows verwendet werden kann. So ist es zum Beispiel möglich, Wissen als Services zur Verfügung zu stellen, sodass mehrere Wissenschaftler dieses Wissen nutzen können. Durch Workflows können innerhalb einer Gruppe die Ergebnisse gemeinsam analysiert werden. Workflows können mit einer großen Menge an Daten umgehen, die bei komplexen Berechnungen notwendig sind. Diese Daten können bei wissenschaftlichen Workflows durch Sensoren oder durch Algorithmen entstehen. Ebenfalls bieten Workflows die Möglichkeit an,

langlebige Prozesse in einer heterogenen und verteilten Umgebung auszuführen. Gerade im wissenschaftlichen Bereich ist das extrem wichtig, da es eine Vielzahl an Plattformen und Programmiersprachen gibt, die genutzt werden. Zusätzlich automatisieren Workflows bestimmte Schritte während der Entwurfs- und Ausführungsphase, sodass sich Wissenschaftler auf die Lösung der Probleme konzentrieren können [GSK⁺₁₁].

Neben den Gemeinsamkeiten gibt es aber auch Unterschiede. Der wissenschaftliche Workflow Lebenszyklus ist auf die Arbeitsweise von Wissenschaftlern zugeschnitten. Ein gravierender Unterschied besteht darin, dass bei wissenschaftlichen Workflows der Personenkreis, der bei einem Workflow mitwirkt, viel kleiner ist, als bei normalen Workflows. In der Regel gibt es einen Wissenschaftler, der den Workflow modelliert, ausführt und analysiert. Es ist natürlich möglich, dass mehrere Wissenschaftler kollaboriert arbeiten, allerdings gibt es keinen Spezialisten für den Entwurf, wie bei Geschäftsprozessen. Dort entwirft ein Spezialist die Workflows. Danach lädt ein IT Spezialist das Modell auf den Server hoch. Ein Angestellter bzw. ein Klient führt den Geschäftsprozess aus und IT und Geschäftsspezialisten analysieren die Ergebnisse. Das alles erledigt bei einem wissenschaftlichen Workflow der einzelne Wissenschaftler. Bei wissenschaftlichen Workflows muss die Möglichkeit geboten werden, dass Wissenschaftler ihre Experimente und Workflows modellieren und sofort starten können. Häufig werden nur Teile eines Experimentes modelliert und diese dann direkt ausgeführt. Ebenfalls soll sich der Wissenschaftler nicht mit technischen Details aufhalten, wie zum Beispiel dem Deployment. Da Wissenschaftler häufig ein Experiment abbrechen und neue Parameter hinzufügen, muss ein wissenschaftlicher Workflow gestoppt und dann an der gestoppten Stelle wieder fortgesetzt werden können. Ebenfalls trennen Wissenschaftler nicht zwischen Modellierung und Ausführung. Das bedeutet, dass aus Sicht des Wissenschaftlers ein Modell gleichzeitig auch eine Instanz ist, im Gegensatz zur gängigen Workflow Technologie, bei der ein Modell mehrere Instanzen haben kann. Jedes Mal, wenn ein neuer Workflow gestartet wird, sehen Wissenschaftler das als neues Experiment an. Ebenfalls unterscheiden sie nicht zwischen der Ausführung und dem Monitoring. Für Wissenschaftler ist es essentiell in Echtzeit Daten über den Workflow präsentiert zu bekommen. So können sie auf eventuelle Anomalien reagieren und den Workflow anhalten und verändern und danach fortsetzen. Der wissenschaftliche Lebenszyklus spiegelt das Konzept **Model-as-you-go** wieder: die Vermischung von Modellierungs-, Ausführungs-, Laufzeitänderungs- und Monitoring Phase [SK₁₀].

2.3. Service Oriented Architecture

Im folgenden Abschnitt wird die Service orientierte Architektur vorgestellt. Es wird eine Definition gegeben und auf die wesentlichen Komponenten der Architektur eingegangen.

2.3.1. Definition

Starke und Tilkov definieren Service Oriented Architecture (SOA) wie folgt [ST₀₇]:

Service Oriented Architecture

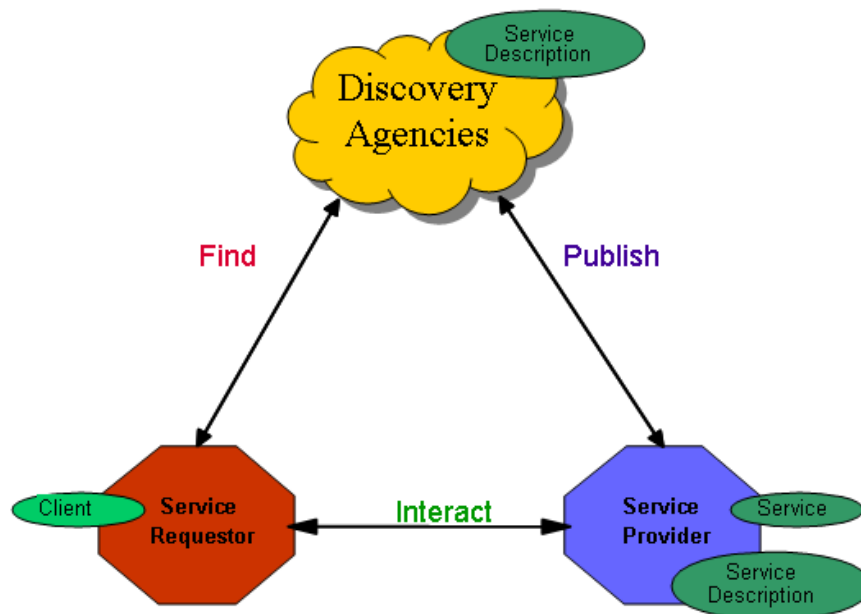


Abbildung 2.9.: SOA Dreieck [CFNO02]

Eine Service orientierte Architektur (SOA) ist eine Unternehmensarchitektur, deren zentrales Konstruktionsprinzip Services (Dienste) sind. Dienste sind klar gegeneinander abgegrenzte und aus betriebswirtschaftlicher Sicht sinnvolle Funktionen. Sie werden entweder von einer Unternehmenseinheit oder durch externe Partner erbracht.

SOA ist ein Architekturstil, bei dem Services die zentrale Rolle spielen. SOA ist das Resultat der Änderungen im Geschäfts- und Technologiebereich. Durch die Notwendigkeit von Outsourcing und dem Reengineering von Geschäftsprozessen, entwickelte sich der Architekturstil [WCL⁺05]. Das SOA Dreieck 2.9 veranschaulicht die Beziehungen zwischen einem Serviceanbieter, einem Servicenutzer und dem Servicerepository.

Der Servicenutzer sendet eine Anfrage nach einem bestimmten Service an das Servicerepository. Dieser schaut in seiner Registrierungsliste nach, ob so ein Service registriert ist. Ein Service wird dann im Repository registriert, wenn ein Anbieter seinen Service veröffentlicht. Das Repository liefert eine geeignete Servicebeschreibung an den Servicenutzer und dieser

bindet sich an den Serviceanbieter. Der Servicenutzer weiß nicht wie die Funktionalität realisiert ist. Er erhält nur eine Schnittstellenbeschreibung des Dienstes.

2.3.2. Web Service

Als Services werden häufig sogenannte Web Services verwendet. Web Services sind nicht die einzige Möglichkeit eine SOA aufzubauen. Es gibt noch andere, wie die Object Management Group - Common Object Request Broker Architecture (OMG CORBA) oder Microsoft Distributed Component Object Model (DCOM). Jedoch haben sich Web Services in der IT etabliert. Das World Wide Web Consortium (W3C) definiert einen Web Services wie folgt [CFNO02]:

A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Service Description Language (WSDL)). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using Hypertext Transfer Protocol (HTTP) with Extensible Markup Language (XML) serialization in conjunction with other Web-related standards.

Somit beschreibt ein Web Service ein Softwaresystem, welches eine Interaktion zwischen zwei Maschinen über das Netzwerk erlaubt, die miteinander genutzt werden können. Sie beschreiben eine Schnittstelle in einer maschinenlesbaren Sprache (Web Service Description Language - WSDL), auf die normalerweise öffentlich zugegriffen werden kann. Andere Systeme können nachrichtenbasiert mit dem Web Service interagieren (meist über SOAP-Nachrichten). Dabei muss die Nachricht der Beschreibung in der WSDL entsprechen. Die Nachrichten sind häufig in XML verfasst und können über unterschiedliche Transportprotokolle versendet werden (z.B. HTTP, JMS).

Die Web Services sind lose gekoppelt und können dynamisch aufgerufen werden. Web Services sind immer verfügbar, also im Stand-By, genau wie Strom bzw. Wasser. Dadurch dass, Web Services durch das W3C standardisiert sind, können unterschiedliche Firmen darauf zugreifen und ihre Services beschreiben [WCL⁺05].

Die Web Service Architektur ist in Abbildung 2.10 dargestellt.

Wie schon erwähnt bieten Web Services Funktionalitäten an, die durch Schnittstellen nach außen hin beschrieben sind. Die Schnittstellen werden in der Regel durch WSDL beschrieben. Ein Auszug einer WSDL Datei befindet sich im Anhang A.1.

SOAP ist eine Nachrichtenarchitektur und ein Verarbeitungsmodell für Nachrichten. Mit Hilfe von SOAP können Nachrichten zwischen Web Services ausgetauscht werden [SS00]. Ein Auszug einer SOAP Nachricht befindet sich im Anhang A.2.

Als Services werden häufig sogenannte Web Services verwendet [WCL⁺05, Gra02]. In dieser Arbeit wird nicht näher auf die Web Service Architektur, auf WSDL und auf SOAP eingegangen, da sie zwar verwendet werden, aber nicht von so großer Bedeutung für das

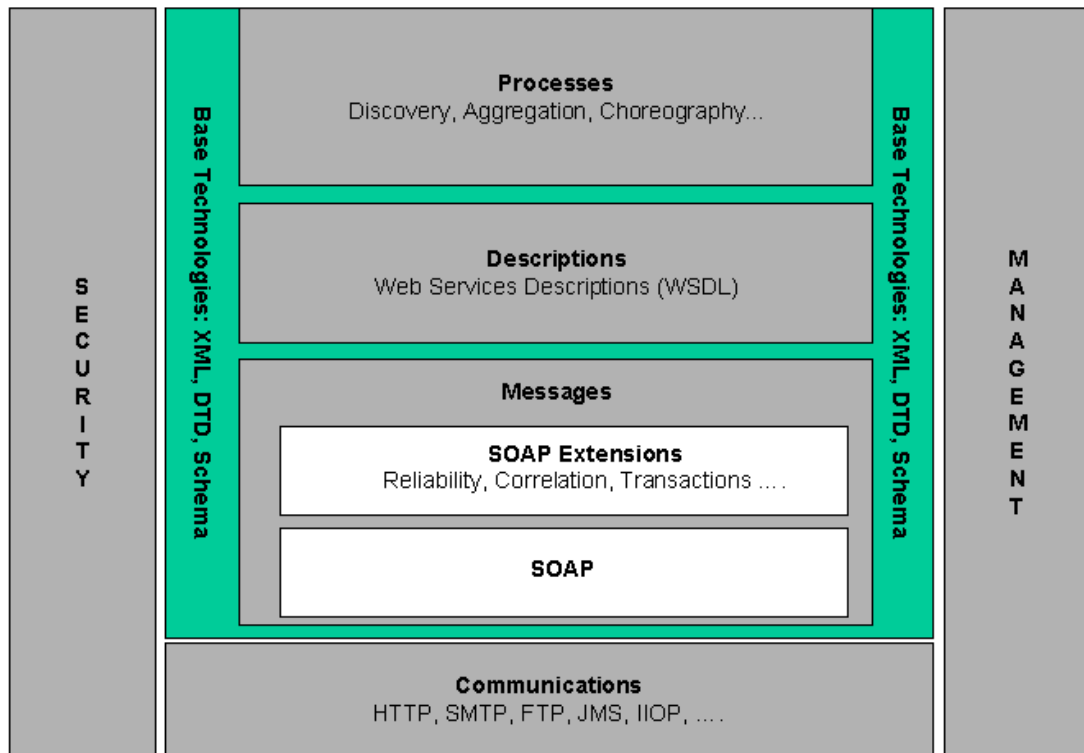


Abbildung 2.10.: Web Service Architektur [CFNO02]

Konzept und die Realisierung sind. Jedoch sollte der Leser wissen was Web Services, WSDL und SOAP sind.

2.4. Business Process Execution Language

Ein Workflow kann, wie oben definiert, eine Reihe unterschiedlicher Aktivitäten besitzen. Solch eine Aktivität kann auch der Aufruf eines Web Services sein. Ein Workflow, welcher als Aktivitätsimplementierung Web Services verwendet, wird als **service composition** bezeichnet, da der Workflow sich aus mehreren Web Services zusammensetzt [GSK⁺11]. Nun bedarf es einer Programmiersprache, die es ermöglicht, solche Workflows zu beschreiben. Außerdem muss die Programmiersprache der Service Orientierung gerecht werden. Punkte wie lose Kopplung, flexible Integration oder Fehlerbehandlung und Kompensation müssen mit der Programmiersprache definiert werden können [WCL⁺05]. Solch eine Programmiersprache existiert und nennt sich **Business Process Execution Language (BPEL)**.

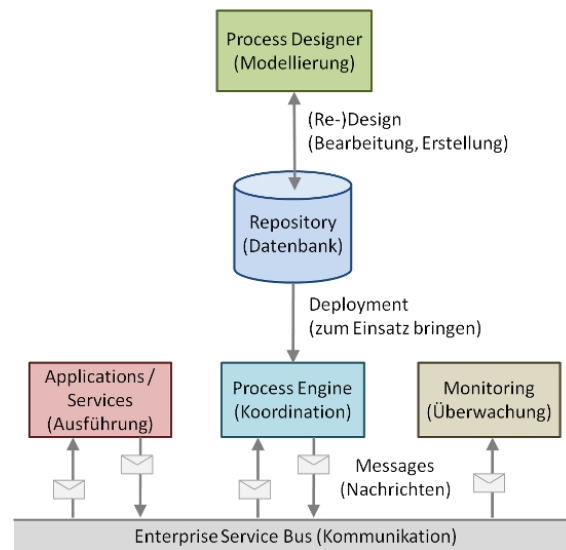


Abbildung 2.11.: BPEL Infrastruktur ^a

^a<http://wirtschaftslexikon.gabler.de/media/910/581440.jpeg>

2.4.1. Definition

Zunächst wird definiert was BPEL konkret ist. Lessen et al. definieren BPEL wie folgt [LLN11]:

Die Business Process Execution Language für Webservices, kurz BPEL oder auch WS-BPEL, ist eine domänenspezifische, imperative Programmiersprache zur Definition von ausführbaren Geschäftsprozessen.

BPEL unterscheidet sich insofern von anderen Programmiersprachen, als dass sie langlaufende, parallele Workflows beschreibt, in denen der Daten- und Kontrollfluss anders spezifiziert werden, als bei normalen Programmen. Dadurch liegt der Fokus auf dem Two-Level-Programming. Die konkrete Implementierung von Services in einer bestimmten Programmiersprache wird als **programming in the small** definiert. Hier wird der konkrete Algorithmus, der Zugriff auf die Daten und die Schnittstelle nach außen implementiert. Die Orchestrierung dieser Services auf einer höheren Abstraktionsebene nennt man **programming in the large** [LR00]. BPEL basiert dabei auf XML.

2.4.2. Infrastruktur

Die BPEL Infrastruktur ist in Abbildung 2.11 dargestellt. Sie besteht aus mehreren Komponenten, die miteinander arbeiten.

Der Prozess Designer kann mit Hilfe eines Modellierungswerkzeugs seinen Prozess modellieren. Es ist möglich, den Prozess auch in einem Texteditor zu beschreiben, aber die gängige Methode ist die grafische Modellierung. Das Modellierungswerkzeug verwaltet sämtliche notwendigen Dateien, wie die BPEL Prozess Datei(en), die WSDL Datei(en) und die XSD Datei(en). Die WSDL Datei ist dabei die Beschreibung der Schnittstelle zur Außenwelt, mit deren Hilfe man den Prozess aufrufen kann. Weiterhin sind die WSDLs der Kommunikationspartner enthalten. Die XSD Datei speichert die im Prozess verwendeten Datentypen. Für diese Arbeit wird der Eclipse BPEL Designer¹ als Modellierungswerkzeug verwendet. Eine konkrete Beschreibung folgt in Abschnitt 2.5. Die genannten Dateien (und in der Regel zusätzlich noch der Deployment Descriptor) werden in ein Paket zusammengefasst und an die BPEL-Engine übermittelt. Die in dieser Arbeit verwendete Engine ist die Apache ODE². Eine genau Beschreibung folgt in Abschnitt 2.5. Diese führt den Prozess dann aus. Dazu wird der Prozess deployt und in der Datenbank gespeichert. Nun wartet der deployte Prozess auf eine Nachricht, die eine Instanz des Prozesses startet. Danach wird die Prozessinstanz nach der Prozessbeschreibung ausgeführt und Zustände und weitere Informationen in der Datenbank abgespeichert, sodass die Monitoring Komponente darauf zugreifen und die Daten visuell präsentieren kann. Dabei können Variablenwerte, Zustände oder das Betreten oder Verlassen einer Aktivität abgespeichert und angezeigt werden. Die Kommunikation mit der Außenwelt läuft über den Enterprise Service Bus ab, der Nachrichten entgegen nimmt und die Serviceaufrufe durch verschiedene Transportprotokolle und ggf. Nachrichtenformate übernimmt [LLN11].

2.4.3. BPEL Modell

Zur Ausführung von Prozessen bietet BPEL ein Modell an. Der Kern des Modells sind geschachtelte **scopes**. Sie beschreiben Relationen zu externen Partnern, Deklarationen von Prozessdaten, verschiedene Handler und Aktivitäten. Der äußerste **scope** ist die Prozessdefinition [WCL⁺05].

Daten in BPEL werden durch typisierte Variablen gespeichert. Die Werte dieser Variablen sind entweder Nachrichten, die zwischen dem Prozess und den Partnern ausgetauscht werden, einfache oder komplexe Datentypen. Die Variablen werden entweder durch WSDL message types, XML Schema types oder XML schema elements definiert. Für die Manipulation der Variablen werden Assign Aktivitäten verwendet. XPath wird als Query-Sprache genutzt, um auf Knoten in einem XML-Baum (hier BPEL Prozess) zu zeigen [Bono8]. Mit Hilfe von XPath kann direkt auf die einzelnen Variablen innerhalb eines Prozesses navigiert werden [WCL⁺05].

Bei den Aktivitäten, die innerhalb eines Prozesses definiert werden, unterscheidet man zwischen strukturierten und atomaren Aktivitäten. Strukturierte Aktivitäten beinhalten weitere Aktivitäten und beschreiben den Kontrollfluss zwischen ihnen. Atomare Aktivitäten

¹<http://www.eclipse.org/bpel/>

²<http://ode.apache.org/>

stehen immer einzeln da und beinhalten keine weitere Aktivitäten, z.B. Aktivitäten zur Manipulation von Variablen oder für die ausgehende und eingehende Interaktion mit einem Web Service.

Folgende Aktivitäten werden als strukturiert bezeichnet [WCL⁺05]:

sequence Aggregation von Aktivitäten, die in der spezifizierten Reihenfolge ausgeführt werden

if Bedingte Verzweigung von Aktivitäten

while Schleifenkonstrukt für Aktivitäten

flow Parallele Ausführung von Aktivitäten

Folgende Aktivitäten werden als atomar bezeichnet:

receive Erhält eine Nachricht von einem Client. Der Client kann ein anderer Prozess oder ein Programm sein. Damit kann auch eine asynchrone Operation definiert werden

reply Sendet eine Nachricht an einen Prozess oder an ein Programm. Damit werden synchrone Operationen beschrieben

invoke Ruft einen anderen Web Service auf. Das kann ein Prozess oder eine Funktionalität sein, die als Web Service definiert ist

assign Kopiert Daten zwischen Variablen und PartnerLinks. Dafür können XPath Ausdrücke verwendet werden

Die beiden Listen sind nicht vollständig. Sie bieten nur einen kleinen Überblick über die vorhandenen Modellelemente. In den Listen sind nur die aufgelistet, die für die Arbeit relevant sind. Entweder werden sie in einem Prozess verwendet oder die Aktivitäten werden für die Arbeit erweitert. Eine komplette Liste befindet sich in der BPEL Spezifikation [Org07].

Wie bereits erwähnt, können Prozesse mit anderen Prozessen kommunizieren und Nachrichten austauschen. Jedoch muss jeder Prozess wissen, mit welchem Prozess bzw. Partner er kommunizieren kann. **Partner Links** definieren solche Partner, damit der Prozess weiß, mit wem er kommunizieren muss. Es ist eine Art Kanal zwischen zwei Partnern, die über eine Point-to-Point Verbindung miteinander kommunizieren [WCL⁺05]. Jeder **partner link** besitzt maximal 2 Rollen, eine für den Prozess selbst und eine für den Partner. Die Schnittstellen des Prozesses bzw. des Partners sind mit der jeweiligen Rolle verknüpft.

Ein Prozessmodell besitzt häufig sehr viele Prozessinstanzen. Um sicher zu stellen, dass ein Partner mit der korrekten Instanz kommuniziert, spielt die Zuordnung eine große Rolle. Da viele Prozessinstanzen parallel laufen, ist es wichtig, dass die korrekte Instanz angesprochen wird. Dafür bietet BPEL einen Korrelationsmechanismus, mit Hilfe der **correlation sets**, an. Die **correlation sets** bestehen aus **properties**, die Instanzen eindeutig definieren, z.B. durch eine **property**, die auf Werte von WSDL Nachrichten gemappt werden. Somit können diese **properties** als Korrelationsmechanismen verwendet werden [WCL⁺05].

Prozesse werden in scopes definiert. Sie bilden dabei einen Block, in dem Aktivitäten ausgeführt werden. Schlägt ein Prozess fehl, kann es sein, dass dabei nur ein einzelner scope fehlgeschlagen ist. Für die Behandlung solcher Fehler existieren für jeden **scope** ein **fault handler** und ein **compensation handler**. Der **fault handler** behandelt Fehler innerhalb eines scopes. Über **compensation handler** können im Fehlerfall bereits erfolgreich abgeschlossene Scopes rückgängig gemacht werden. Auf diese Weise können mit BPEL Transaktionen realisiert werden. Es gibt noch weitere Handler, die alle in der Spezifikation aufgelistet sind [Org07].

2.5. Verwendete Werkzeuge

Für die Modellierung von Workflows existieren zahlreiche Programme. Für diese Arbeit werden zur Bearbeitung und Ausführung von Workflows der Eclipse BPEL Designer als Modellierungswerkzeug und die Apache ODE als Workflow-Engine verwendet. Im Folgenden wird ein kurzer Überblick über beide Werkzeuge gegeben.

2.5.1. Eclipse BPEL Designer

Der Eclipse BPEL Designer ist ein Open-Source Projekt unter der Leitung des Eclipse Technology Project³. Das Projekt verfolgt das Ziel, innerhalb der Eclipse Umgebung BPEL Prozesse definieren, bearbeiten, deployen, testen und debuggen zu können. Zusätzlich soll das Projekt erweiterbar sein, damit Unternehmen ihre eigenen Wünsche schnell in das Projekt integrieren können.

Für diese Arbeit wird der Eclipse BPEL Designer von SimTech der Universität Stuttgart verwendet. Der Eclipse BPEL Designer wird für wissenschaftliche Workflows erweitert, um den Anforderungen gerecht zu werden.

In dieser Arbeit werden Eclipse BPEL Designer, BPEL Editor und BPEL Designer als Synonyme verwendet.

Im Folgenden werden die Hauptbestandteile des Eclipse BPEL Designers aufgelistet und kurz erklärt. Die Auflistung bezieht sich auf die Diplomarbeit von [Eic10]:

Model Das Modell repräsentiert die Spezifikation BPEL 2.0 und steht als Eclipse Modeling Framework (EMF)⁴-Modell zur Verfügung

Validation Der Validator arbeitet auf dem EMF-Modell und gibt Fehler und Warnungen aus, wenn es Probleme in Bezug auf die Spezifikation gibt. Dadurch wird die Korrektheit des Modells sichergestellt

³<http://www.eclipse.org/technology/>

⁴<http://www.eclipse.org/modeling/emf/>

Runtime Framework Ein erweiterbares Framework, das es erlaubt, einen Prozess in eine BPEL-Engine zu deployen und dort auszuführen. Bisher existiert eine Unterstützung der Apache ODE

Designer Der BPEL Designer ist ein Graphical Editing Framework (GEF)⁵-basierter Editor, der als Hilfsmittel zum Schreiben von BPEL-Prozessmodellen dient

Debug Ein Framework, das es dem Benutzer erlaubt, die Ausführung des Prozesses zu verfolgen und auch Breakpoints anbietet. Dieser Teil wurde noch nicht implementiert

Implementiert wird das BPEL Project in verschiedenen Plugin-Projekten, von denen die wichtigsten hier erwähnt werden:

org.eclipse.bpel.common.model Implementiert wird hier ein Framework für ein EMF-Modell

org.eclipse.bpel.common.ui Hier werden allgemeine Klassen für ein Graphical User Interface (GUI) implementiert, die in org.eclipse.bpel.ui verwendet werden

org.eclipse.bpel.model In diesem Projekt enthalten sind das EMF-Modell und die daraus generierten Klassen des BPEL-Modells

org.eclipse.bpel.ui Hier befindet sich die Implementierung des Editors für BPEL Prozessmodelle

org.eclipse.bpel.runtimes Implementiert wird hier ein Framework, welches Unterstützung für das Deployment und die Ausführung in einer BPEL-Engine liefert

org.eclipse.bpel.validator In diesem Projekt wird der Validator implementiert

org.eclipse.bpel.apache.ode.runtime Hier wird eine direkte Integration mit der Apache ODE implementiert. Es handelt sich um eine Referenzimplementierung und es wird das Framework aus org.eclipse.bpel.runtimes genutzt

Der BPEL Designer basiert auf einem EMF-Modell. EMF steht für Eclipse Modeling Framework und ist ebenfalls ein Projekt des Eclipse Technology Project. EMF basiert auf einem strukturierten Datenmodell, von dem aus Modelle und Code generiert werden. Ausgehend von einer Modellspezifikation in XML Metadata Interchange (XMI), bietet EMF Werkzeuge und Laufzeitunterstützung zur Generierung von Java Modellklassen an. Zusätzlich werden eine Reihe von Adaptern generiert, die es ermöglichen, Modelle visuell zu präsentieren und mit Hilfe von Commands zu bearbeiten. Bei der Generierung werden immer ein Interface und eine Implementierung des Interfaces erstellt. Das EMF besteht aus drei wichtigen Bestandteilen [SBPM09].

Der erste Bestandteil ist der EMF Core. Dieser bietet ein Metamodell zur Beschreibung der Modelle und Unterstützung zur Laufzeit an. Die Laufzeitunterstützung beinhaltet die

⁵<http://www.eclipse.org/gef/>

Benachrichtigung von Änderungen am Modell, die Persistierung eines Modells mit XMI Serialisierung und eine API zur Bearbeitung von EMF Objekten.

Der zweite Bestandteil ist das EMF Edit Framework. Es beinhaltet generische und wiederverwendbare Klassen zur Erstellung von Editoren für EMF-Modelle. Zusätzlich können noch weitere Hilfsklassen, z.B. Contentprovider oder Labelprovider Klassen erzeugt werden. Außerdem beinhaltet das EMF Edit Framework noch ein Command Framework, das wiederum eine Menge an generischen Command Klassen besitzt, um Editoren zu erstellen, die automatisch Undo und Redo bereitstellen.

Der letzte Bestandteil ist das EMF Codegen. Damit ist es möglich, Modellklassen aus einer Modellspezifikation zu generieren. Mit Hilfe einer Benutzeroberfläche können die notwendigen Einstellungen übernommen und die Klassen generiert werden.

2.5.2. Apache ODE

Wenn ein Workflow definiert wird, sollte dieser auch ausgeführt werden. Das geschieht in einer Engine. Auch hier gibt es zahlreiche Engines. Da der Eclipse BPEL Designer eine Referenzimplementierung der Apache ODE anbietet, wird für diese Arbeit auch diese Engine verwendet. Im Folgenden wird auf die Architektur eingegangen, um einen groben Überblick zu geben.

Der Hintergrund für die Entwicklung der Apache ODE bestand darin, eine zuverlässige, kompakte und integrierte Komponente zu schaffen, die langlebige BPEL Prozesse ausführen kann. Die lose Kopplung war zur Erstellung eines vollwertigen Business Process Management System von höchster Priorität [ODE]. Die Hauptkomponenten der Apache ODE Architektur sind in Abbildung 2.12 dargestellt.

Es wird ein BPEL Dokument durch den Compiler in eine Form gebracht, welche die Runtime ausführen kann. Die Runtime führt dann den Prozess zuverlässig aus und persistiert diesen über Data Access Objects. Der Integration Layer verhält sich wie eine Zwischenschicht, welche die Engine mit der Außenwelt verbindet und Daten nach außen reicht. Im Folgenden werden die einzelnen Komponenten kurz erläutert [ODE]:

ODE BPEL Compiler Der BPEL Compiler ist dafür verantwortlich, das BPEL Dokument und die dazugehörigen Dokumente (WSDL und Schemata) in kompilierte Repräsentanten umzuwandeln, sodass die Runtime diese zur Ausführung verwenden kann. Dabei wird ein Objektmodell erzeugt, welches dem BPEL Dokument ähnlich ist

ODE BPEL Engine Runtime Die Runtime führt den kompilierten Prozess aus. Dafür stellt es Implementierungen der einzelnen BPEL Konstrukte bereit. Außerdem ist die Runtime zuständig, neue Instanzen zu erstellen und Nachrichten an die korrekten Instanzen weiter zu leiten. Zusätzlich implementiert die Runtime die Management API, mit deren Hilfe ein Benutzer mit der Engine interagieren kann. Um zuverlässig arbeiten zu können, werden die Konstrukte persistiert. Dafür werden die Data Access Objects verwendet. Durch das ODE Java Concurrent Object Framework (Jacob) ist es möglich,

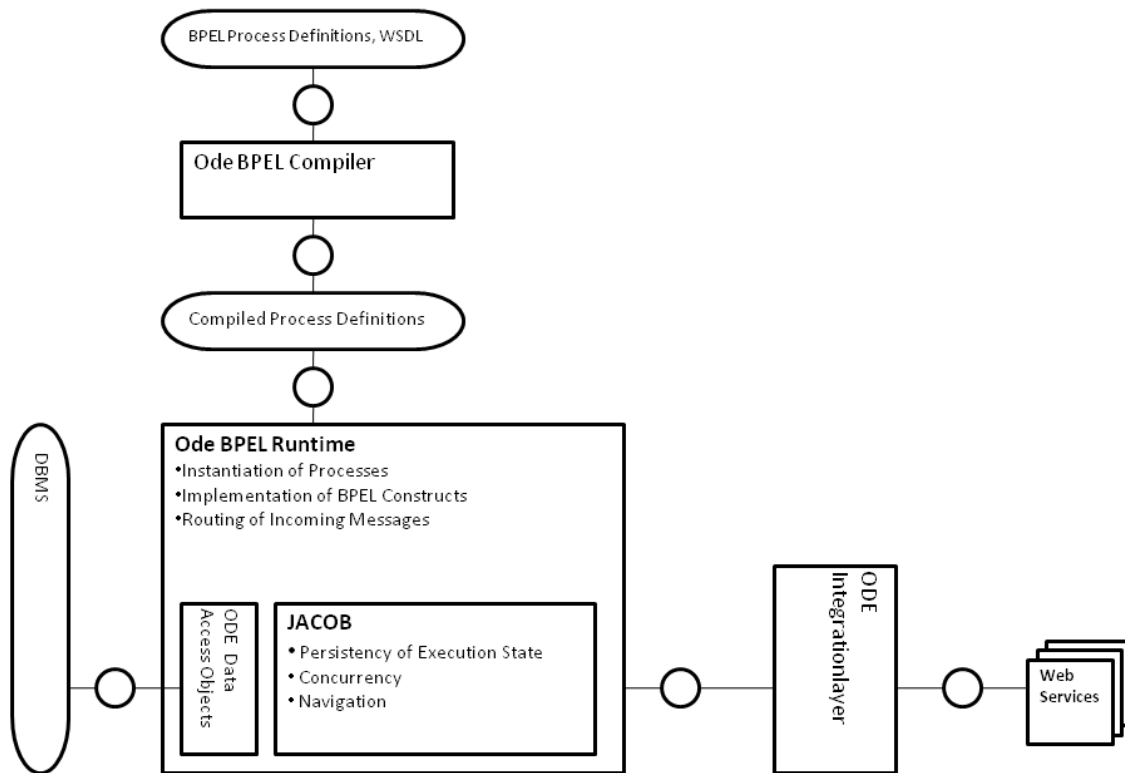


Abbildung 2.12.: Apache ODE Architektur [ODE]

auf Instanzebene mehrere Instanzen parallel laufen zu lassen. Außerdem bietet Jacob einen Mechanismus zum Unterbrechen einer laufenden Instanz und das Persistieren des Zustandes an. Somit werden alle BPEL Konstrukte auf Instanzebene durch Jacob abgewickelt. Eine konkrete Beschreibung findet sich in der Beschreibung zur ODE's virtuellen Maschine [ODE]

ODE Data Access Objects (DAO) DAOs sind die Verbindungsstellen zwischen der Runtime und einer darunter liegenden Datenbank. Hierbei beruht die Persistierung auf OpenJPA, wobei auch Hibernate möglich ist. Durch die Möglichkeit der Persistierung können aktive Instanzen gespeichert, Nachrichten an die korrekten Instanzen, die darauf warten, weiter geleitet, Variablen und Partner Links verändert und Zustände gespeichert werden

ODE Integration Layers (ILs) Die Runtime kann nicht alleine existieren. Sie kann nicht mit der Außenwelt kommunizieren und ihre Daten nach außen weiter reichen. Der Integration Layer integriert daher die Runtime in eine Umgebung, die mit der Außenwelt kommunizieren kann, z.B. Axis2⁶ oder Java Business Integration (JBI). Der Integration

⁶<http://axis.apache.org/axis2/java/core/>

Layer bietet Kommunikationskanäle zur Außenwelt an. Der IL kommuniziert bei Axis2 durch Web Service Interaktion und bei JBI durch das Binden der Runtime in den JBI Nachrichten Bus

2.5.3. BPEL Event Modell

Basierend auf [KKS⁺06] wird folgend ein Engine unabhängiges BPEL Event Modell beschrieben. Es wird kurz auf die Funktionsweise und anschließend ebenfalls kurz auf die Realisierung des Event Modells auf der Apache ODE und beim BPEL Designer eingegangen.

Das Event Modell von [KKS⁺06] bezieht sich auf die BPEL Spezifikation 1.1. In der Diplomarbeit von Thomas Steinmetz [Steo8] wird dieses Event Modell auf die BPEL Spezifikation 2.0 angepasst. Falls von einem Event Modell in dieser Arbeit gesprochen wird, bezieht es sich auf das für die BPEL Spezifikation 2.0.

Das Event Modell beschreibt Events, die innerhalb des Lebenszyklus eines Prozesses, einer Aktivität, eines Scopes, einer Schleife oder eines Links auftreten. Die Events werden von der BPEL Engine erzeugt und zeigen die Zustandsübergänge an. Diese Events müssen nach außen weiter gereicht werden, um externe Anwendungen über die Zustandsübergänge zu benachrichtigen [KKS⁺06].

Wie bereits erwähnt, werden Events für Prozesse, Aktivitäten, Scopes, Schleifen oder Links erzeugt und abgesendet. Es besteht jedoch die Möglichkeit Events zu erzeugen, wenn von außen eine Nachricht in die Engine ankommt. So kann auf externe Events, wie von einem Monitoringprogramm oder einer anderen BPEL Engine, reagiert werden [KKS⁺06]. In der Abbildung 2.13 sind die Zustandsübergänge eines Prozesses abgebildet.

Process_Deployed Dieses Event wird gefeuert, wenn ein neuer BPEL-Prozess auf einer BPEL Engine deployt wird. Dieses Event wird nicht für jede Instanz gefeuert, sondern für jedes Prozessmodell

Process_Instantiated Dieses Event wird gefeuert, wenn ein Prozessmodell durch ein Receive oder Pick Aktivität instantiiert wird

Instance_Running Dieses Event wird gefeuert, wenn eine Instanz läuft, nachdem sie instantiiert oder fortgesetzt wurde

Instance_Suspended Dieses Event wird gefeuert, wenn eine Instanz, durch einen Breakpoint oder einer externen Anfrage, unterbrochen wird

Instance_Terminated Dieses Event wird gefeuert, wenn eine Instanz durch die <terminate> Aktivität beendet wird

Instance_Complete Dieses Event wird gefeuert, wenn ein Prozess erfolgreich beendet wird, d.h. ohne Fehler und ohne Beendigung durch <terminate>

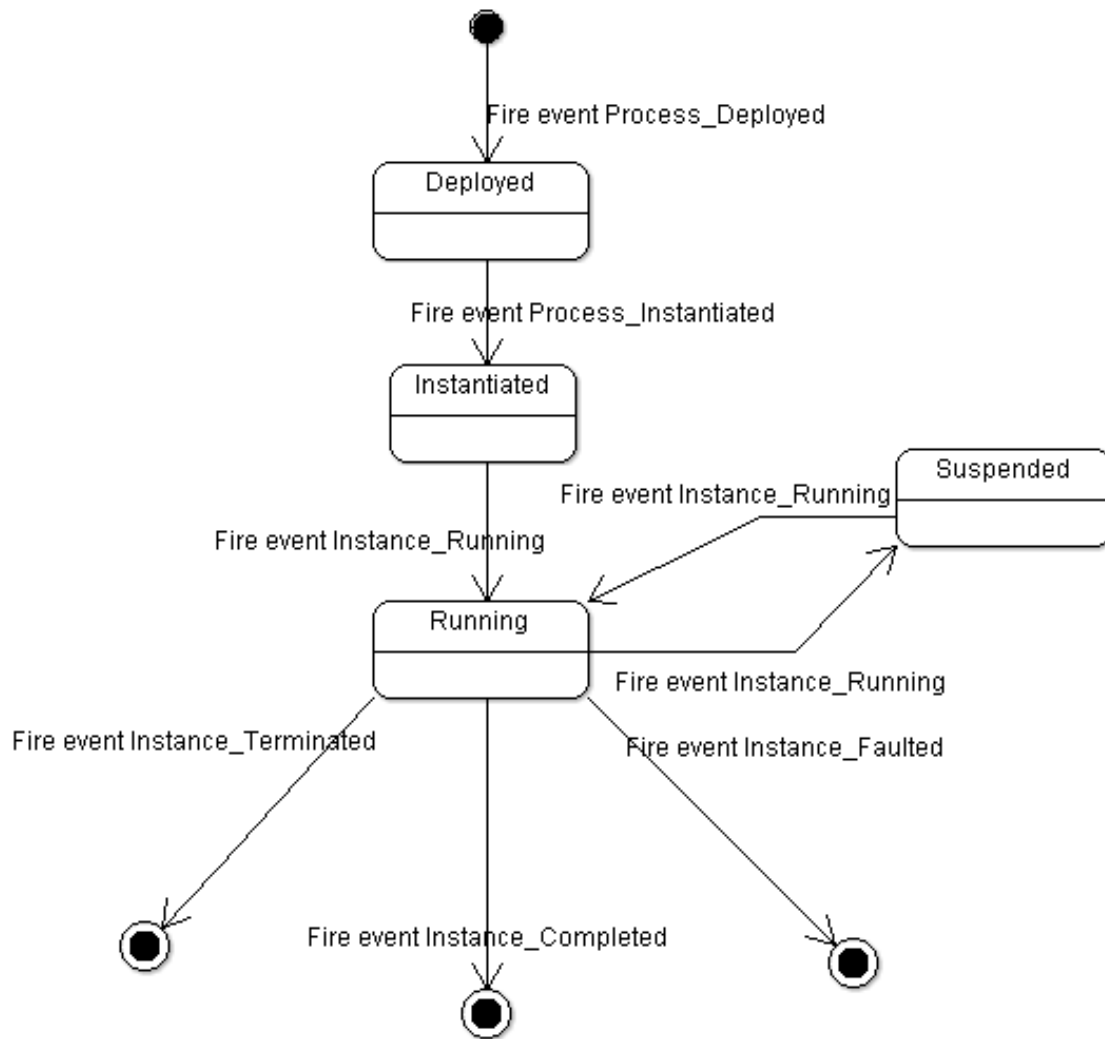


Abbildung 2.13.: Zustandsdiagramm eines Prozesses [KKS⁺06]

Instance_Faulted Dieses Event wird gefeuert, wenn eine Instanz durch einen Fehler beendet wird, der nicht behandelt wird und an den Fault-Handler des root-scope des Prozesses weiter geleitet wird

Eine vollständige Auflistung der Zustandsdiagramme des Event Modells befindet sich in [KKS⁺o6].

Da das Event Modell ein Engine unabhängiges Modell ist, muss es für die einzelnen Engines speziell realisiert werden. Da die freie BPEL Engine Apache ODE in dieser Arbeit verwendet wird, wird nun kurz auf die Realisierung des Modells innerhalb der Apache ODE eingegangen.

Nachdem ein Event auf der Engine gefeuert wird, werden für sie zuständige Handler aufgerufen. Diese Handler bearbeiten die Events weiter. So wird zunächst unterschieden, ob das Event durch die Zustandsänderung einer Aktivität innerhalb einer Instanz oder durch die Zustandsänderung der Instanz selber gefeuert wurde. Die Engine bietet auch einen Handler für ankommende Nachrichten an. Dadurch ist es möglich, dass externe Anwendungen Events auf der Engine feuern können. Die jeweiligen Handler bieten Methoden an, um mit den entsprechenden Events weiter zu arbeiten. Nachdem die Events abgearbeitet sind, werden Nachrichten mit dem dazugehörigen Typ erstellt und an den Client gesendet. Wird z.B. auf dem Client ein Variablenwert geändert, kann mit Hilfe eines Events eine Nachricht generiert werden, die an die Engine geschickt wird. Dort erhält die Engine die Information, dass der Variablenwert einer Variable geändert werden soll. Die entsprechenden Handler ändern dann den Variablenwert. Die Nachricht besitzt alle nötigen Informationen für die korrekte Korrelation.

Damit die Events, die von der Engine erzeugt und gefeuert werden, auf der Client Seite verarbeitet werden können, bietet der Eclipse BPEL Designer einige Klassen an, die zur Verwaltung der Events beitragen. Sind die Nachrichten am Client angekommen, wird anhand des Nachrichtentyps die Nachricht an den jeweiligen korrekten Manager weitergeleitet. Der Manager verarbeitet dann die Nachricht entsprechend. Es existieren mehrere Manager. Zunächst werden sie vom Engine Output Event Handler an den jeweiligen Manager (Instance-, Activity-, Variable Manager) weiter gesendet. Diese Manager beziehen sich, wie auch bei der Engine, auf Aktivitäten und auf Instanzen. Der BPEL Designer bietet noch die Möglichkeit, Nachrichten an die Engine zu senden und somit bestimmte Requests zu erzeugen. Bei einer Antwort eines Requests bearbeitet der BPEL Designer diese Antworten ebenfalls. Eine ausführliche Beschreibung bietet die Diplomarbeit von Eichel [Eic10].

3. Verwandte Arbeiten

Es gibt bereits einige Arbeiten über parametrisierte Workflows und über das Monitoring. Außerdem existieren Modellierungswerkzeuge, die solche Funktionalitäten anbieten. In diesem Kapitel soll auf die verwandten Arbeiten hingewiesen und beschrieben werden, welchen Nutzen sie für diese Arbeit haben.

3.1. Entwicklung eines Monitoring-Tools zur Unterstützung von parametrisierten Web Service Flows

In [Nito6] wurde ein Monitoringwerkzeug für BPEL Prozesse realisiert. Es wurde eine eigenständige Anwendung entwickelt, die das Monitoring übernimmt. Diese Diplomarbeit ist die Grundlage für die Realisierung der Monitoring Funktion in der Diplomarbeit von [Eic10]. Basierend auf ihr wurde die Monitoring Funktion im Eclipse BPEL Designer eingebaut, allerdings nicht als eigenständige Anwendung, sondern integriert. ActiveBPEL wurde als Engine verwendet. Das Monitoring von Nitzsche bezieht sich auf wirtschaftliche Workflows und nicht auf wissenschaftliche Workflows.

3.2. Interaktives Monitoring von wissenschaftlichen Workflows

In [Eic10] wurde eine Monitoring Funktion für den Eclipse BPEL Designer realisiert. Diese Realisierung bildet die Grundlage für die Aufgabe in dieser Arbeit. Jedoch bezieht sich das Monitoring von Eichel nur auf eine einzelne Prozessinstanz. Die Arbeit baut auf [Nito6] auf. Im Rahmen der Erweiterung des Eclipse BPEL Designer in dieser Arbeit wird das Monitoring von Eichel verwendet und dahingehend erweitert, dass das Monitoring für mehrere, parallel laufende Instanzen möglich ist.

3.3. Ein Event Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE

In [Steo8] wurde das im Kapitel 2 vorgestellte BPEL Event Modell realisiert. In der Diplomarbeit wird das vorhandene BPEL Event Modell von [KKS⁺06] erweitert. Dieses Modell bezieht sich nur auf die BPEL Spezifikation 1.1. Das Modell von Steinmetz erweitert das Event Modell, damit es mit der BPEL Spezifikation 2.0 konform ist. Das Event Modell wird

in dieser Arbeit um eine Verarbeitung eines Events erweitert, die notwendig ist, um parallel mehrere Instanzen überwachen zu können.

3.4. A Monitoring Tool for the Apache ODE BPEL Engine

In [Utko8] wurde ein Monitoringwerkzeug für BPEL Prozesse realisiert. Die Arbeit baut auf [Nito6] auf. Im Gegensatz zu Nitzsche realisiert Utkin das Monitoringwerkzeug für die BPEL Engine Apache ODE. Die Architektur von [Nito6] wird komplett übernommen. Nur technologische Unterschiede der beiden Engines werden angepasst.

3.5. Integration of the Result Visualization into a Workflow Modeling Tool

In [Cheo9] wurde eine Visualisierung von Simulationsergebnissen mit Hilfe eines Visualisierungsworkflows realisiert. Das Hauptaugenmerk liegt auf der Eingabe von bestimmten Parametern im Modellierungswerkzeug, die für die Visualisierung der Simulationsergebnisse verwendet werden. Die Parameter werden vom Benutzer eingegeben und an die Engine geschickt, auf der der Visualisierungsworkflow deployt ist. Der Workflow wird von der Engine ausgeführt und liefert das erzeugte Bild zurück an das Modellierungswerkzeug, wo es schließlich angezeigt wird. Der in [Cheo9] verwendete Prozess wird für diese Arbeit im Anwendungsbeispiel verwendet.

3.6. Sedna

Es gibt eine Reihe von Werkzeugen, die sich mit wissenschaftlichen Workflows beschäftigen und die Möglichkeit bieten, diese zu erstellen und auszuführen.

Sedna ist ein Projekt der Open Middleware Infrastructure Institute UK (OMII)¹ [WEB⁺07]. Im Rahmen des Projektes wurde ein Eclipse-Plugin zur Modellierung von wissenschaftlichen Workflows erstellt². OMII-BPEL ist dabei die Realisierung von Sedna, die mit Hilfe der freien Workflow-Engine ActiveBPEL wissenschaftliche Workflows ausführt. Sedna beruht auf der geschäftlichen Workflow Technologie und bietet neben der Modellierung auch die Validierung und die Erweiterung des Modellierungswerkzeugs durch Plugins an. OMII-BPEL ist dabei nur eine Realisierung. Sedna bietet ein Konzept zur Nutzung unterschiedlicher Workflow-Engines. Sedna übersetzt und exportiert die unterschiedlichen Sprachelemente in BPEL und erstellt Deployment Descriptors für unterschiedliche Workflow-Engines.

¹<http://www.omii.ac.uk/>

²<http://sse.cs.ucl.ac.uk/index.php?id=omii-bpel>

3.7. Microsoft Trident

Microsoft Trident ist ein Modellierungswerkzeug von Microsoft, welches auf die herkömmliche Workflow Technologie aufbaut [BJA⁺07]. Es nutzt die Microsoft Workflow Foundation als Technologieplattform und verwendet eine kontrollgesteuerte Modelliersprache, die auf die Extensible Orchestration Markup Language (XOML) aufbaut. Trident besitzt einzelne Komponenten zur Modellierung und Ausführung von Workflows. So können Wissenschaftler z.B. aus mehreren Modellierungswerkzeugen auswählen. Diese Modellierungswerkzeuge können ein Text oder ein graphischer Editor sein [GSK⁺11]. Ebenfalls bietet Trident die Möglichkeit, gestartete Workflows zu überwachen und sie mit unterschiedlichen Parametern zu starten.

Es gibt noch eine Reihe weiterer wissenschaftliche Modellierungswerkzeuge, die hier im Folgenden erwähnt werden, jedoch nicht weiter beschrieben werden, da sie für diese Arbeit nicht relevant sind. Kepler, wird hauptsächlich von Bioinformatikern verwendet [ABJ⁺04]. Pegasus verlagert komplexe Berechnungen in eine Grid, indem es die abstrakte Workflow Beschreibung auf die vorhandenen Grid Ressourcen abbildet und den Workflow dann ausführt [DBG⁺04]. Taverna wird bevorzugt von Biologen verwendet [OGA⁺06]. Es spezialisiert sich auf langlebige Workflows, die Daten getrieben sind. Triana konzentriert sich auf die Ausführung in verteilten Umgebungen [TSWH07]. In einem Szenario, in dem drei unterschiedlich Arten von Workflows genutzt werden, wird Triana erläutert [CGH⁺06]. Der erste Workflow ist ein wissenschaftlicher Workflow zur Datenverarbeitung von Gravitationswellensignalen. Der zweite Workflow ist eine Auftragserteilung, welches Triana Services in einer Testumgebung ausführt. Der letzte Workflow beschreibt einen Monitoring Workflow, der Änderungen an einer laufenden Anwendung untersucht und ausführt.

4. Anforderungen

In diesem Kapitel werden unabhängig von den technischen Voraussetzungen, die Anforderungen an diese Arbeit erläutert. Dadurch besteht die Möglichkeit, die Realisierung auf mehreren Plattformen durchzuführen und der Leser erhält einen Überblick darüber, was in dieser Arbeit entwickelt wird.

4.1. Rahmenbedingungen

Da Wissenschaftler explorativ arbeiten und häufig in einem Experiment ihre Parameter ändern, soll das Modellierungswerkzeug und die Engine den unten stehenden Anforderungen gerecht werden. Zudem arbeiten Wissenschaftler häufig an mehreren Experimenten gleichzeitig. Diese spiegeln sich in mehreren Workflows wieder. Dabei muss der Wissenschaftler sämtliche Workflows im Blick haben. Dafür ist es notwendig, dass das Modellierungswerkzeug dieser Anforderung gerecht wird. Aus diesem Grund müssen das Modellierungswerkzeug und die Workflow-Engine dementsprechend erweitert werden.

4.2. Anforderungen

- Der Wissenschaftler soll die Möglichkeit haben, bestimmte Variablen als Parametervariablen zu definieren. Parametervariablen sind besondere Variablen, die in einem Experiment festgelegte Werte aus einem bestimmten Wertebereich annehmen können. Die Werte werden vom Wissenschaftler spezifiziert. Durchlaufen die Parameterwerte gar den Wertebereich oder einen Teil davon, spricht man von Parameterstudien. Parameterstudien werden im wissenschaftlichen Umfeld genutzt, um Parameterwerte oder -kombinationen herauszufinden, die zu einem gesuchten Phänomen führen
- Dem Wissenschaftler soll eine grafische Oberfläche zur Verfügung gestellt werden, mit dessen Hilfe er die Werte der zuvor definierten Parametervariablen eingeben kann
- Die eingegebenen Werte der Parametervariablen sollen den Body einer Nachricht bilden, die an die Engine versendet wird. Diese Nachricht startet dann den entsprechenden Workflow
- Auf der Engine Seite soll diese Nachricht dahingehend verarbeitet werden, dass die Werte der Nachricht ausgelesen und an die entsprechende Stelle im Workflow, den Variablen zugewiesen werden

4. Anforderungen

- Der Wissenschaftler soll sich nicht mit technischem Wissen aufhalten. Er soll nur seine Werte eingeben müssen. Den Rest erledigen die Engine und der Client
- Parallel laufende Workflows sollen zur selben Zeit verfolgt und beobachtet werden können. Das Monitoring muss für alle laufende Instanzen vorhanden sein
- Der Client soll Informationen einer Instanz anzeigen, die zum momentan ausgewählten Prozessmodell zugehörig ist
- Der Client soll Informationen zu gespeicherte Instanzen anzeigen können. Dabei sollen für alle ausgewählten Instanzen die Informationen angezeigt und korrekt zugeordnet werden

5. Konzept

In diesem Kapitel soll das Konzept dieser Arbeit vorgestellt werden. Dabei gliedert sich das Konzept in zwei Teile. Die beiden Konzepte sind unabhängig voneinander entwickelt worden und stehen nicht miteinander in Verbindung. Zunächst wird das Konzept für das parametrisierte Starten von Workflows vorgestellt, anschließend das Konzept für die visuelle Korrelation von parallelen Workflows. Es wird auch Bezug auf die bestehende Architektur genommen, damit der Leser einen Überblick erhält, was bereits vorhanden ist und was neu konzipiert werden musste.

5.1. Parametrisiertes Starten von Workflows

Wie bereits in Kapitel 4 beschrieben, soll das Starten der Workflows parametrisiert erfolgen, da Wissenschaftler explorativ arbeiten und häufig Werte ändern müssen. Daher soll dem Wissenschaftler die Möglichkeit geboten werden, in einer Oberfläche diese Werte eingeben zu können und dann damit den Workflow zu starten.

5.1.1. Ist-Zustand

Durch bereits vorhergehende Diplomarbeiten wurden einige Aspekte eines Modellierungswerkzeugs erweitert, um den Anforderungen an wissenschaftlichen Workflows gerecht zu werden. In Abbildung 5.1 wird das momentane Konzept für den Aufruf der wissenschaftlichen Workflows im Sequenzdiagramm dargestellt.

Zunächst wird im Modellierungswerkzeug der Prozess modelliert. Nach dem Modellieren wird eine Nachricht erstellt. Die Nachricht ist eine Dummy-Nachricht, die keine relevanten Informationen enthält. Die Nachricht dient nur dazu, um den Workflow, der auf der Workflow-Engine auf eine Nachricht wartet, anzustoßen und zu signalisieren, dass von außen der Workflow gestartet werden soll. Die Dummy-Nachricht genügt den Anforderungen der explorativen Arbeitsweise der Wissenschaftler nicht. Daher muss die Nachricht und der Aufruf der Workflows dementsprechend verändert werden, sodass die Nachricht die nötigen Informationen beinhaltet, um die Workflows parametrisiert starten zu können. Mit dem start Befehl wird der Workflow-Engine signalisiert, den Prozess zu starten und auszuführen. Die Engine arbeitet alle Aktivitäten innerhalb des Prozesses ab und sendet Events bei Zustandsänderungen an das Modellierungswerkzeug, die zur weiteren Verarbeitung verwendet werden. Die Dummy-Nachricht hat zur Folge, dass Parameter, die der Wissenschaftler spezifiziert, hart in den Workflow codiert werden müssen, sodass die Werte nicht flexibel

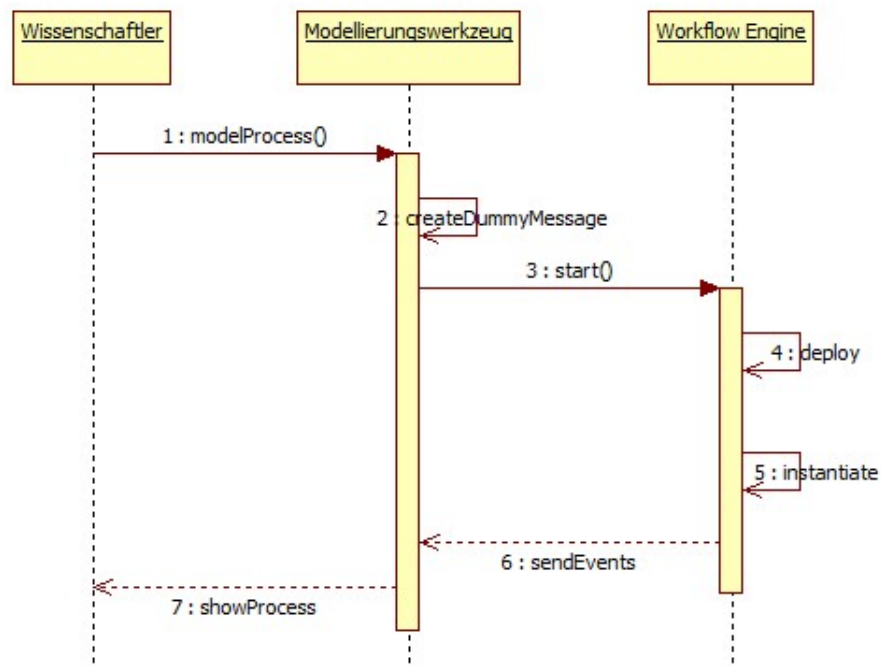


Abbildung 5.1.: Starten eines Workflows [Eic10]

und dynamisch eingetragen werden können. Außerdem lassen sich keine Parameterstudien durchführen, indem eine Reihe von unterschiedlichen Parametern spezifiziert wird und mit ihnen Workflows gestartet werden. Das Konzept zur Realisierung der parametrisierten Ausführung eines wissenschaftlichen Workflows wird im nächsten Abschnitt dargestellt.

5.1.2. Konzept für die Parametrisierung

Die explorative Arbeitsweise der Wissenschaftler erfordert einen Mechanismus zum Starten der Workflows, die es ermöglicht, bestimmte Parameter beim Start an die Engine zu übergeben. Wie bereits angesprochen, wird momentan ein Workflow mit einer Dummy-Nachricht aufgerufen und gestartet, die keine relevanten Informationen enthält. Zusätzlich kann ein Wissenschaftler momentan seine gewünschten Parameter nicht eingeben, sodass ein Workflow immer mit gleichen Werten gestartet wird.

Die Abbildung 5.2 stellt als Sequenzdiagramm den Ablauf eines parametrisierten Aufrufs eines Workflows dar.

Im neuen Konzept modelliert der Wissenschaftler zunächst seinen Prozess. Beim Starten des Prozesses wird nun nicht, wie im alten Konzept, direkt eine Nachricht erstellt und an die Workflow-Engine gesendet, sondern es wird zunächst ein Parameterdialog geöffnet.

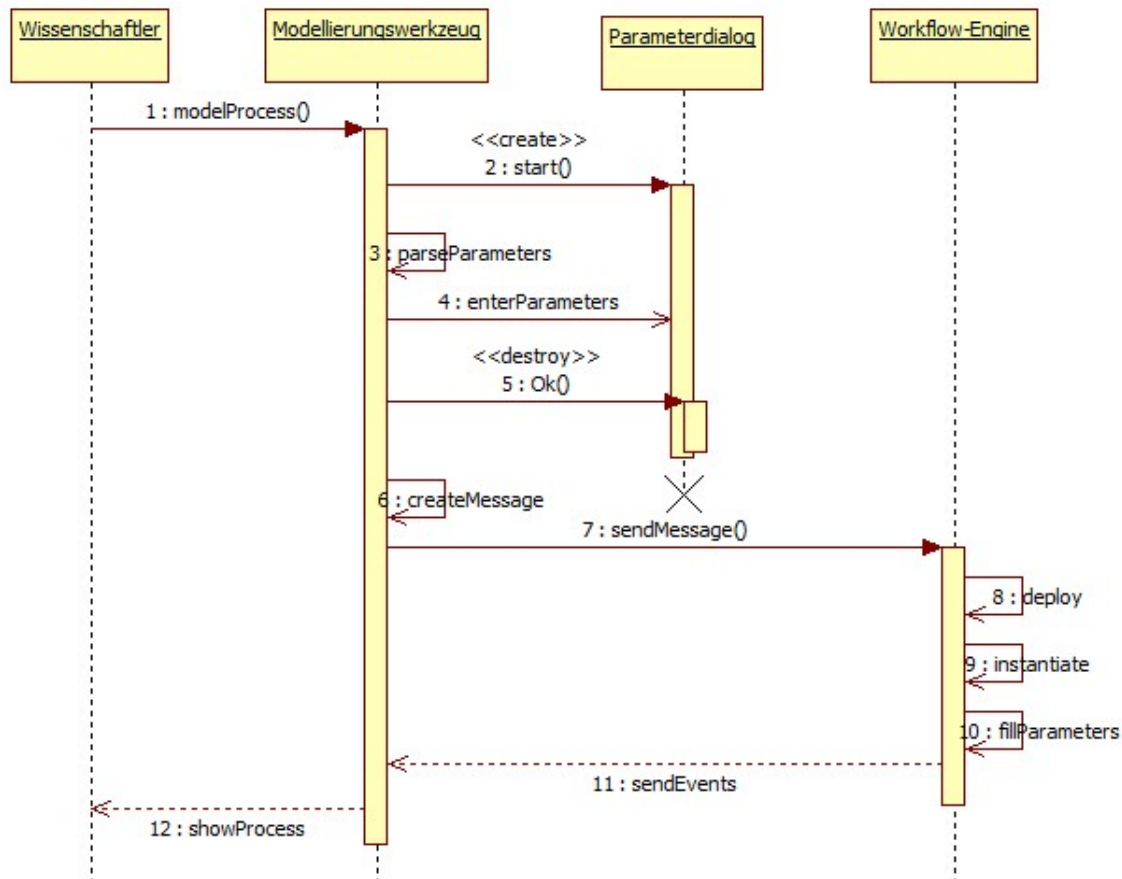


Abbildung 5.2.: Parametrisierter Aufruf eines Workflows

Mit Hilfe dieses Parameterdialogs, der die zuvor definierten Parametervariablen anzeigt, die im Prozess markiert sind, kann der Wissenschaftler Parameterwerte eintragen. Die Parameterwerte werden in eine Nachricht verpackt und an die Engine gesendet. Die Engine deployt und instantiiert den Workflow und kopiert die Parameterwerte, die sich in der Nachricht befinden, in die Variablen. Die einzelnen Komponenten im Sequenzdiagramm werden im Folgenden beschrieben. Die Abbildung 5.3 veranschaulicht die Abhängigkeiten der einzelnen Komponenten.

Parsen des Prozesses

Der Wissenschaftler kann im Modellierungswerkzeug sein Prozessmodell modellieren und gestalten. Nachdem der Wissenschaftler mit der Modellierung fertig ist, startet der Wissenschaftler den Workflow. Bevor der Prozess jedoch auf der Engine deployt wird, wird der

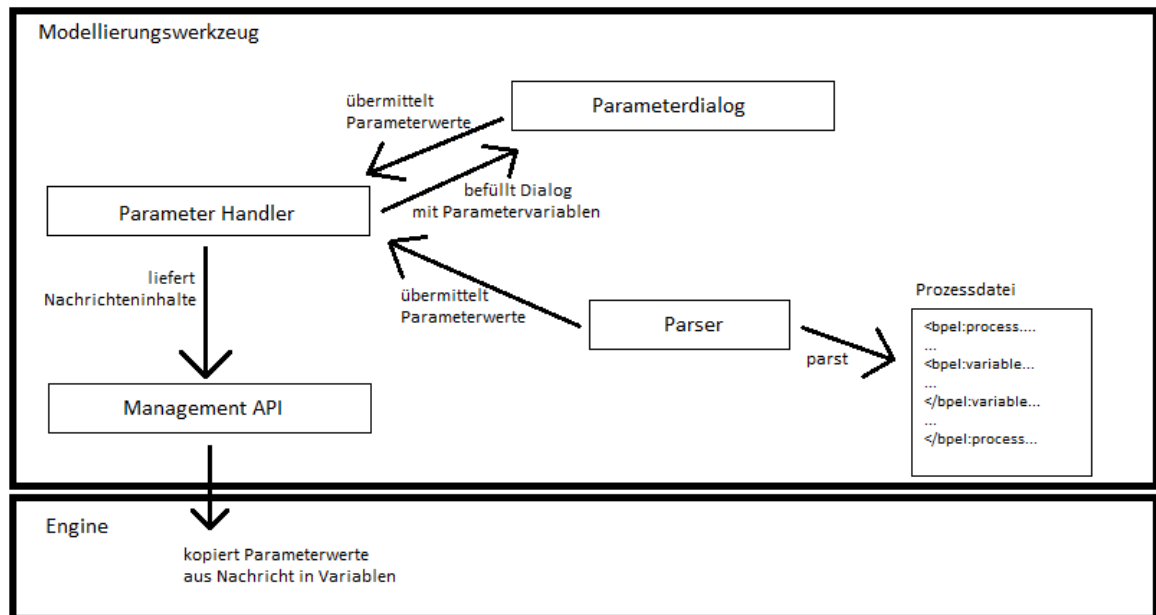


Abbildung 5.3.: Darstellung des Ablaufs für parametrisiertes Starten

Prozess nach Variablen geparkt. Da es in BPEL verschachtelte Scopes gibt und Variablen zu unterschiedlichen Scopes angehören, müssen beim Parsen die Variablen auch ihren korrekten Scopes zugeordnet werden. Somit wird sichergestellt, dass beim Kopieren der Parameterwerte, die richtige Variable im richtigen Scope gewählt wird. Zuvor kann der Wissenschaftler Variablen als Parametervariablen markieren, die vom Parser erkannt werden. Die Markierung der Variablen führt zu einer Erweiterung des BPEL Schemas im Modellierungswerkzeug. Damit der Wissenschaftler die Variablen markieren kann, wird dem Variable Element des Schemas ein weiteres Attribut hinzugefügt: das **parameter** Attribut. Es ist vom Typ Boolean und speichert, ob eine Variable als Parametervariable gekennzeichnet ist oder nicht. Das Parsen des Prozesses erfolgt bei jedem Start eines Workflows.

Parameter Handler

Nachdem der Prozess geparkt ist und alle Parametervariablen identifiziert sind, müssen diese Parametervariablen abgespeichert werden, damit sie im Dialog angezeigt werden können und der Wissenschaftler seine Werte eintragen kann. Für die Speicherung der Parametervariablen musste eine Entwurfsentscheidung gefällt werden, da zwei Möglichkeiten existieren, die Parametervariablen zu speichern. Zum Einen können die Parametervariablen in eine Datenbank dauerhaft abgelegt und von dort immer wieder aufgerufen werden, zum anderen werden sie in eine Liste gespeichert, sodass sie transient sind. Für diese Arbeit werden die Parametervariablen in eine Liste abgespeichert. Diese Art der Speicherung wurde deshalb gewählt, da der Aufwand für die Speicherung in eine Datenbank keinen Mehrwert

Listing 5.1 Schema der Nachricht, die an die Workflow-Engine gesendet wird

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="ParameterRequest">
    <xs:complexType>
      <xs:element name="parameters" type="xs:parameters"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="parameters">
    <xs:sequence>
      <xs:element name="varName" type="xs:any"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

enthält und das Parsen des Prozesses sich nicht auf die Performanz auswirkt, da das Parsen der Parametervariablen nur einen Bruchteil einer Sekunde benötigt, sodass bei jedem Start eines Workflows das Modell geparkt werden kann und die Parametervariablen in die Liste abgespeichert werden können. Die Liste, zur Speicherung der Parametervariablen, stellt der sogenannte Parameter Handler zur Verfügung. Der Parameter Handler ist auch die Schaltzentrale zwischen dem Parser und dem Dialog und ist ebenfalls für die Zusammensetzung des Dialogs zuständig. Er hält alle gespeicherten Parametervariablen mit seinen Werten und bietet diese Informationen dem Dialog an, damit der Dialog die korrekten Parametervariablen anzeigt. Zusätzlich kümmert sich der Parameter Handler um das Befüllen der WSDL Datei. Die WSDL Datei wird dahin gehend verändert, dass sie im **types** Abschnitt neue Elemente in der Sequenz erhält. Diese Elemente enthalten den Variablennamen als Elementnamen und als Typ den Variablentyp. Der **types** Abschnitt beschreibt die Form der Nachricht, die erstellt werden muss, um den Workflow auf der Engine starten zu können. Später wird mit Hilfe der WSDL Datei die Nachricht erstellt. Die Nachricht besteht aus allen Parametervariablen mit den dazugehörigen Werten. Listing 5.1 zeigt das XML-Schema der Nachricht, die aus der WSDL Datei gebaut und an die Engine gesendet wird. VarName ist der jeweilige Name der Parametervariable.

Parameterdialog

Im Parameterdialog werden die geparkten Parametervariablen angezeigt. Je nachdem welcher Datentyp gewählt wurde, baut der Parameter Handler den Dialog zusammen. Dieser bietet Methoden an, um den jeweils korrekten Eintrag aufzubauen.

Der Wissenschaftler kann nun seine Werte eintragen. Je nach Datentyp sind eine oder mehrere Angaben möglich. So soll der Wissenschaftler nicht nur einzelne Zahlenwerte, sondern auch ganze Zahlenbereiche angeben können. Das gleiche gilt für Strings. Hier sind mehrere Eingaben möglich. Dadurch, dass nicht nur einzelne Werte, sondern ganze Bereiche

mit Schrittweiten angegeben werden können, müssen daraus einzelne Nachrichten gebaut werden. Konkret heißt das, es muss das kartesische Produkt aus den Eingaben erstellt werden. Das kartesische Produkt beschreibt nach Merziger alle geordneten Paare (a,b) zweier Mengen A und B , wobei a aus A und b aus B ist [MWo6]. Dabei wird das kartesische Produkt als $A \times B$ geschrieben. Als Formel wird das kartesische Produkt folgendermaßen beschrieben:

$$A \times B := \{(a,b) | a \in A, b \in B\}$$

Eine Verallgemeinerung ist das kartesische Produkt von n Mengen $A_1, \dots, A_n$, wobei es aus allen n -Tupeln $a_1, \dots, a_n$ mit a_i aus A_i besteht und man schreibt es als $A_1 \times \dots \times A_n$ oder als $\prod_{i=1}^n A_i = A_1 \times \dots \times A_n := \{(a_1, \dots, a_n) | a_i \in A_i, i = 1, \dots, n\}$.

Für jedes Tupel des kartesischen Produkts wird eine Nachricht gebaut und an die Engine gesendet. Somit werden so viele Workflows gestartet, wie Tupel des kartesischen Produkts existieren. Dadurch lassen sich Parameterstudien durchführen. Beispiele für Nachrichten, die aus dem kartesischem Produkt aufgebaut werden befinden sich im Anhang A.2.

Workflow-Engine

Nachdem dann auf der Engine Seite der Start des Workflows angestoßen wird, müssen die Informationen aus der Nachricht extrahiert und in den Prozess eingefügt werden. Konkret heißt das, die Werte der Parametervariablen müssen als Werte der Variablen in den laufenden Workflow eingesetzt werden. Für das Einsetzen der Werte in den laufenden Workflow existieren zwei Möglichkeiten. Da in BPEL der Datenaustausch durch Nachrichten erfolgt und daraus die Werte von einer Variablen in eine andere kopiert werden (Assign), könnte man beim Starten des Workflows eine Assign-Aktivität ausführen, die alle Werte an die richtige Stelle einfügt. Dieses Assign könnte für den Wissenschaftler unsichtbar gehalten werden, sodass dieser die Aktivität gar nicht bemerkt. Bei diesem Ansatz müsste die Apache ODE nicht verändert werden. Allerdings wird dieser Ansatz nicht weiter verfolgt, da dieser Ansatz die Prozesslogik des Workflows verändert, da eine unsichtbare Assign-Aktivität hinzugefügt wird, die nur die Engine kennt. Der Aufwand wäre zu hoch für die Realisierung, da jedes Mal das Assign geändert bzw. gelöscht werden müsste, wenn der Wissenschaftler eine Variable als Parameter deklariert bzw. löscht. Wird das zur Laufzeit gemacht, wie es beim **Model-as-you-go** Ansatz durchaus möglich ist, d.h. es werden neue Aktivitäten mit neuen Variablen hinzugefügt, dann wäre eine Änderung des Assigns gar nicht mehr möglich, da das Assign bereits ausgeführt wurde.

Der zweite Ansatz richtet sich mehr an das bereits bestehende Konzept der Engine. Da beim Start eines Workflows immer die Receive/Pick-Aktivität aufgerufen und dort die ankommende Nachricht abgespeichert wird, bietet es sich an, an dieser Stelle die Werte der Parametervariablen in den Workflow einzusetzen. Ein normales Receive/Pick speichert die ankommende Nachricht ab und startet den Workflow. Konkret bedeutet das, dass beim Receive/Pick die Initialisierung der Variablen und das Einsetzen der Werte stattfindet. Die Abbildung 5.4 stellt den normalen Ablauf eines Receives/Pick dar.

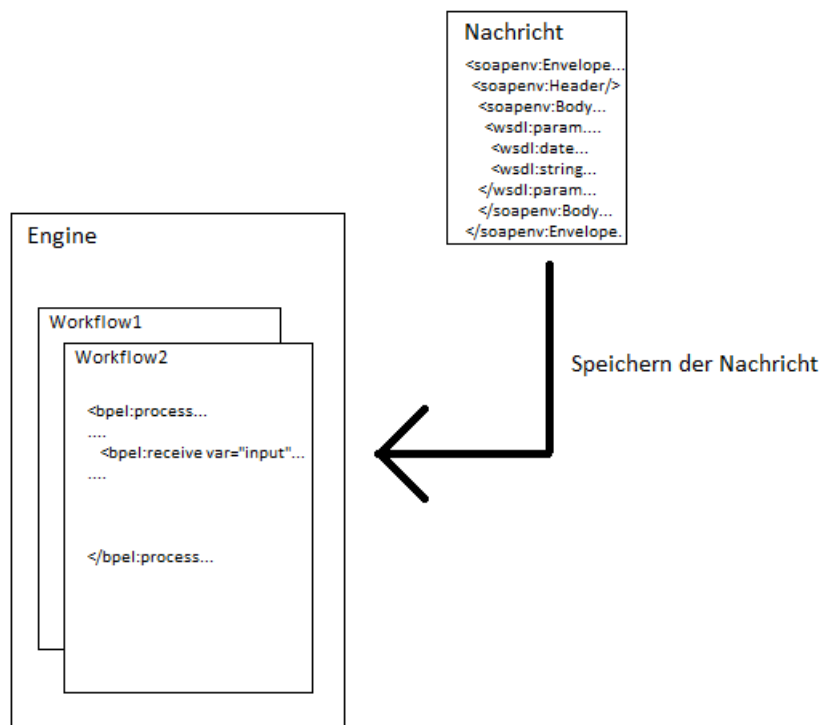


Abbildung 5.4.: Normaler Ablauf einer Receive/Pick Aktivität

Im neuen Konzept wird das Speichern der ankommenden Nachricht nicht verändert. Allerdings werden nun zusätzlich aus der Nachricht die einzelnen Parameterwerte ausgelesen. Die ausgelesenen Parameterwerte werden dann in die richtigen Variablen im Workflow kopiert. Das Receive/Pick bearbeitet nur Parameterwerte, wenn welche vorhanden sind, ansonsten verhält es sich wie ein normales Receive/Pick. Somit sind die Werte für die Variablen beim Starten der Workflow-Instanz vorhanden und können verwendet werden. Die Speicherung der Parameterwerte funktioniert nur beim **parameterRequest** Nachrichtentyp. Der vollständige Name des Nachrichtentyps setzt sich aus dem Prozessnamen und **parameterRequest** zusammen. Dieser wird im Zuge dieser Arbeit erstellt. Bei anderen Nachrichtentypen verhält sich das Receive/Pick normal und es wird nur die Nachricht abgespeichert, aber keine Parameterwerte. Die Abbildung 5.5 stellt das neue Konzept der Receive/Pick Aktivität dar.

Ein Problem stellen die Scopes in BPEL dar. Wie schon beschrieben arbeitet BPEL mit Scopes. In diesen Scopes sind die Variablen abgelegt. Der äußerste Scope ist der Prozess-Scope. Somit sind zunächst nur die globalen Variablen, die im Prozess-Scope definiert werden, sichtbar. Variablen in geschachtelten Scopes sind nicht von außen sichtbar. Somit wird zunächst jeder Scope durchgegangen, um die zu initialisierenden Variablen zu identifizieren. Dafür wird ein bottom-up Vorgehen gewählt, da es unendlich viele geschachtelte Scopes geben kann. Der Prozess-Scope ist der äußerste Scope innerhalb eines Prozesses und bildet somit die äußerste Hülle eines Prozesses. Ist man am Prozess-Scope angelangt ist die Bearbeitung

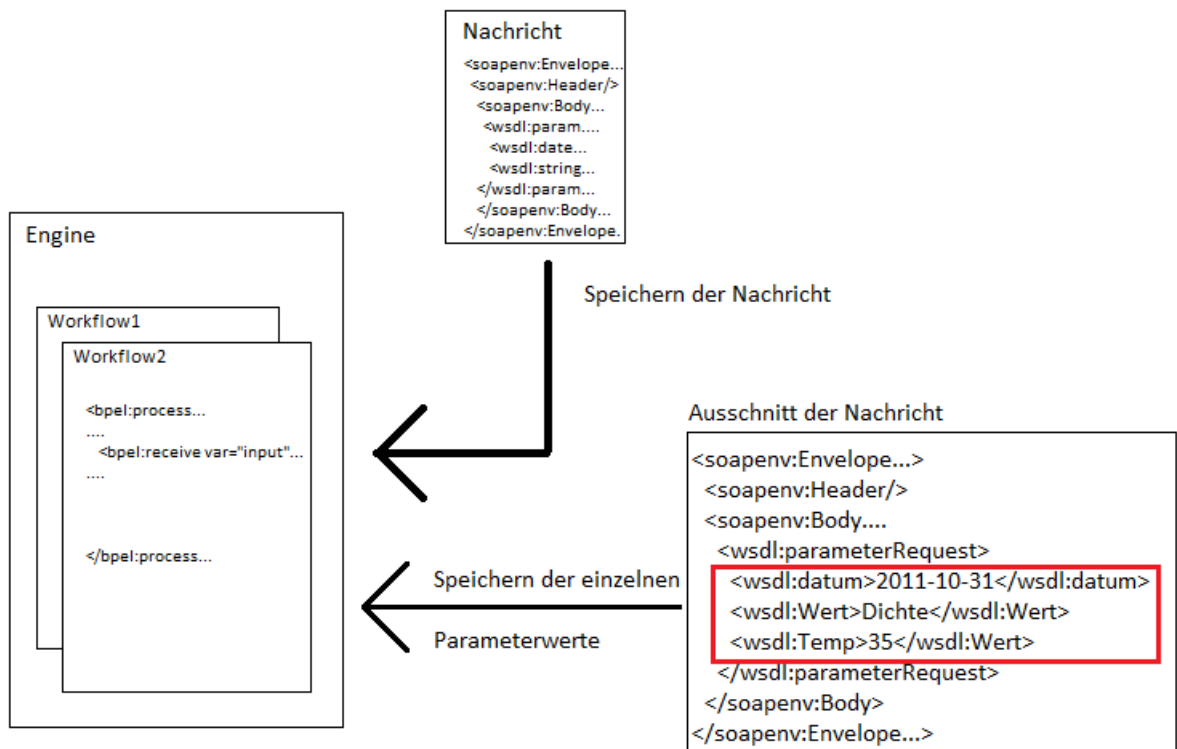


Abbildung 5.5.: Ablauf einer Receive/Pick Aktivität bei Speicherung der Parameterwerte

bei einem bottom-up Vorgehen abgeschlossen und alle zu initialisierenden Variablen sind identifiziert. In dieser Arbeit wird dieser Ansatz gewählt. Es erfordert zwar eine Änderung der Engine, allerdings wird die Initialisierung an der dafür vorgesehen Stelle durchgeführt und es muss nicht die Logik des Workflows geändert werden.

5.2. Visuelle Korrelation von parallelen Workflows

Wissenschaftler müssen häufig mehrere Workflows parallel laufen lassen. Zudem wollen sie noch Informationen zu laufenden Workflows erhalten. Jedoch müssen die Ausführungsevents, die die Engine veröffentlicht, auf Seite des Modellierungs-/Monitoringtools so gefiltert, gehalten und korreliert werden, dass eine saubere Anzeige der einzelnen Workflow Instanzen auch für viele Workflow-Modelle möglich ist. Daher soll nun ein Konzept vorgestellt werden, welches es ermöglicht, parallele Workflows zu überwachen.

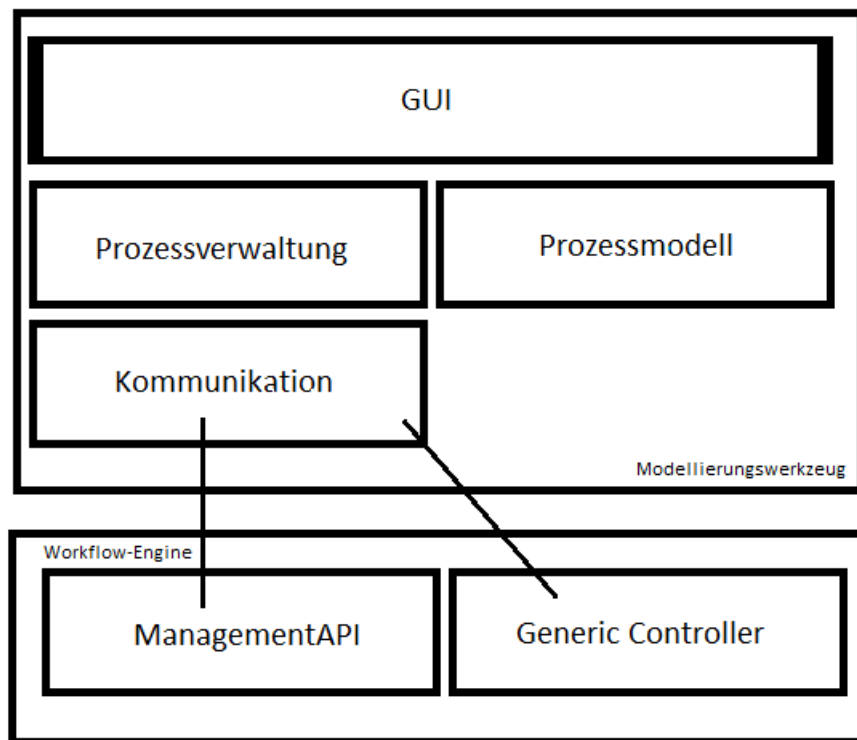


Abbildung 5.6.: Architektur des Monitoringtools [Eic10]

5.2.1. Architektur

Für den zweiten Teil des Konzepts zur visuellen Korrelation von parallelen Workflows wird die bestehende Architektur des Monitoringtools verwendet, die in Abbildung 5.6 dargestellt wird [Eic10].

Die Architektur beschreibt drei Schichten: Kommunikation, Prozessverwaltung und GUI. Das Prozessmodell ist bereits vorhanden und wird in die Architektur aufgenommen, da das Prozessmodell die BPEL Prozesse widerspiegelt. Im Folgenden werden die einzelnen Komponenten beschrieben.

Prozessmodell Das Prozessmodell ist die Repräsentation der BPEL-Spezifikation in Form eines EMF-Modells

Kommunikation In der Kommunikationskomponente werden Nachrichten von der Engine empfangen und an sie gesendet. Einerseits bedeutet dies die Nutzung der Management API, andererseits implementiert sie den Custom Controller. Die Informationen aus Nachrichten werden an die Prozessverwaltung weitergegeben und von dort erfolgt auch die Beauftragung zum Versand von Nachrichten

Prozessverwaltung Die Prozessverwaltung enthält die eigentliche Programmlogik. Sie interagiert mit GUI, Kommunikation und Prozessmodell

GUI Die Benutzeroberfläche erweitert grafische Komponenten des Modellierungswerkzeugs und fügt neue hinzu.

Eine ausführliche Beschreibung bietet die Diplomarbeit von [Eic10]. In dieser Arbeit werden nur die relevanten Aspekte der Diplomarbeit erwähnt und erläutert.

5.2.2. Monitor Manager und Monitoring Provider

Die Architektur von [Eic10] wird weiterhin verwendet und im Zuge dieser Arbeit auch nicht verändert. Basierend auf dieser Architektur hat Eichel das Monitoring im Eclipse BPEL Designer realisiert [Eic10]. Der Monitor Manager spielt eine essentielle Rolle. Der Monitor Manager, welcher der Prozessverwaltungs-komponente angehört, ist dafür zuständig den Status des Monitoring zu kennen und vom Benutzer gewünschte Änderungen des Status durchzuführen. Der Monitor Manager kann momentan nur einen einzigen Workflow überwachen. Da nun Wissenschaftler aber mehrere Workflows parallel laufen lassen sollen und bereits auch können, sollte ebenfalls die Möglichkeit geboten werden, dass der Wissenschaftler mehrere Workflows gleichzeitig überwachen kann. Um dieser Anforderung gerecht zu werden muss der Monitor Manager erweitert werden, sodass die Überwachung mehrerer Workflows möglich ist. Damit für diese Arbeit mehrere Workflows parallel beobachtet werden können, muss genau zugeordnet werden, in welchem Zustand sich eine Instanz gerade befindet. Diese Verwaltung übernimmt der Monitoring Provider, der im Zuge dieser Arbeit erstellt wurde. Er enthält alle gestarteten Instanzen. Jede Instanz besitzt seinen eigenen Monitor Manager. Nun ist die Möglichkeit geboten, dass jede Instanz unabhängig von anderen Instanzen überwacht werden kann. Jeweils die Instanz (Instance Information) und der dazugehörige Monitor Manager werden im Monitoring Provider abgespeichert und anderen Komponenten zur Verfügung gestellt.

5.2.3. Weitere Manager

Da weitere Manager existieren, die für die Bearbeitung einer Instanz notwendig sind, müssen diese ebenfalls angepasst werden. Da zuvor die Architektur darauf ausgelegt war, nur eine Instanz zu überwachen, sind alle Manager darauf ausgelegt nur einen einzigen Workflow zu bearbeiten. Da jede Instanz seinen eigenen Monitor Manager besitzt, müssen die bestehenden Manager genau wissen, um welche Instanz es sich gerade handelt. Der Activity Manager kümmert sich um Events rund um Aktivitäten. Hier erhält der Activity Manager den zuständigen Monitor Manager mit. An diesem können dann die konkreten Zustandsänderungen übertragen werden. Der Instance Manager behandelt Events, die gefeuert werden, wenn Zustandsänderungen einer Instanz stattfinden. Der Instance Manager wird dahingehend verändert, dass er den Monitoring Provider fragt, um welche Instanz es sich handelt. Die korrekte Instanz wird über die Instanz ID festgelegt, die im Event vorhanden ist. Der Variable Manager behandelt alle Events, die gefeuert werden, wenn Änderungen an Variablen

stattfinden. Hier wird der gleiche Ansatz wie beim Activity Manager verfolgt. Der Variable Manager erhält den konkreten Monitor Manager und übermittelt ihm sämtliche Änderungen. Es ist durchaus möglich, dass alle drei Manager den Monitoring Provider fragen, um welche Instanz es sich handelt. Jedoch soll der Fokus der Manager nur auf die Bearbeitung der Änderungen der Aktivität, Variable oder der Instanz liegen und der Monitor Manager soll die restlichen Aufgaben erledigen. Ein weiterer Grund warum der Activity und Variable Manager den entsprechenden Monitor Manager kennen, liegt darin, dass bei der Bearbeitung der Aktivitäten und der Variablen das jeweilige Prozessmodell benötigt wird, also das Modell, das im Modellierungswerkzeug angezeigt wird. Der Monitor Manager beinhaltet dieses Prozessmodell und somit kann am richtigen Prozessmodell gearbeitet werden, um zum Beispiel Zustandsänderungen im Prozessmodell abzuspeichern. Beim Instance Manager wird nichts am Prozessmodell verändert und daher ist es nicht notwendig, dass der Instance Manager den dazugehörigen Monitor Manager kennt. Wird er dennoch benötigt, kann durch die Instanz am Monitoring Provider angefragt werden, welcher Monitor Manager der Instanz angehört.

5.2.4. XPathMapper und XPathMap

Wie bereits erwähnt, werden durch den Activity und Variable Manager Änderungen am Prozessmodell durchgeführt. Wenn sich zum Beispiel der Zustand einer Aktivität ändert, wird das im Modellierungswerkzeug durch den Activity Manager bearbeitet und angezeigt. Der XPathMapper ist dafür zuständig, um an die richtige Stelle im Prozessmodell zu gelangen. Anhand eines XPath Ausdruckes, aus dem Event, wird die korrekte Aktivität ausgewählt und dementsprechend markiert. Es ist möglich im Modellierungswerkzeug mehrere unterschiedliche Prozessmodelle anzuzeigen, jedoch können von einem Prozessmodell mehrere Instanzen existieren. Da der Wissenschaftler auswählen kann, welche Instanz er gerade überwachen möchte, muss zum einen die richtige Instanz und zum anderen das richtige Prozessmodell gewählt werden. Daher erhält der XPathMapper immer den konkreten Monitor Manager der Instanz, die gerade überwacht wird. Somit werden die Informationen der Instanz auf das Prozessmodell, welches der Monitor Manager hält, projiziert, das der Wissenschaftler angefordert hat. Die XPathMap hält alle XPath Ausdrücke eines Prozesses. Bisher wurde nur ein Workflow überwacht. Die ganzen XPath Ausdrücke wurden nur für diesen einen Workflow abgespeichert. Nun müssen mehrere Workflows überwacht und somit auch die ganzen XPath Ausdrücke der unterschiedlichen Workflows gespeichert werden. Somit muss ein Mechanismus geboten werden, der zu einem Prozess alle XPath Ausdrücke liefert. Das geschieht indem die XPath Ausdrücke nun mit dem jeweiligen Prozessmodell in der XPathMap abgespeichert werden.

5.2.5. Auditing

Das Modellierungswerkzeug bietet eine Auditing Sicht an, die alle Events auflistet, die von der Engine gefeuert werden. Da auch diese Sicht darauf ausgelegt ist, nur die Informationen einer Instanz anzuzeigen, muss diese Sicht angepasst werden, sodass immer die Events

der gerade gewählten Instanz angezeigt werden. Die nötigen Informationen sind bereits durch das Auditing Tool vorhanden, welches als eine Standalone Anwendung, im Rahmen von SimTech, realisiert wurde. Allerdings werden nicht alle nötigen Informationen an das Modellierungswerkzeug übertragen. Das Auditing Tool empfängt Nachrichten, die gesendet werden, wenn Events auf der Engine gefeuert werden. Diese Nachrichten werden je nach Typ bearbeitet und die nötigen Informationen in einer Datenbank abgespeichert. Die Auditing Sicht muss dementsprechend geändert werden, dass auch sie mit mehreren Workflows umgehen kann. Ein Korrelationsmechanismus existiert hier nicht, der die Eventinformation der Instanz dem jeweiligen Prozessmodell zuordnet. Das bedeutet, dass im Zuge dieser Arbeit ein Korrelationsmechanismus erstellt wird, der es ermöglicht, bei Auswahl eines Prozessmodells im Modellierungswerkzeug, alle Eventinformationen der dazugehörigen Instanz anzuzeigen. Momentan werden die Events der zuletzt gestarteten Instanz angezeigt.

Damit nun die Auditing Sicht angepasst wird, muss die Instance History Message erweitert werden, sodass alle nötigen Informationen, wie der Eventtyp, der Zeitstempel, der XPath-Ausdruck und der Zustand der Aktivität in der Nachricht enthalten sind. Die Datenbank muss zusätzlich die nötigen Informationen bereit halten. Daher müssen die korrekten Daten erst ausgelesen werden. Im Zuge dieser Arbeit wird dieses Auslesen realisiert und die Erweiterung der Instance History Message realisiert.

Das Konzept bezieht sich nur für das Modellierungswerkzeug. Wenn Workflows innerhalb des Modellierungswerkzeugs gestartet werden, ist die Monitoring und Auditing Funktion verfügbar. Die Funktionen beschränken sich nur auf das Modellierungswerkzeug. Wird ein Workflow über einen anderen Client gestartet, haben die Monitoring und Auditing Funktion keine Wirkung und es ist nicht möglich über das Modellierungswerkzeug Workflows zu überwachen und zu verfolgen. Beim Starten eines Workflows über einen anderen Client werden die Events auf der Engine ausgelöst, allerdings können sie nicht an das Modellierungswerkzeug gesendet werden, da das Modellierungswerkzeug zu diesem Zeitpunkt der Engine unbekannt ist und der Nachrichtenaustausch zwischen Modellierungswerkzeug und Engine nicht funktioniert. Erst beim Starten eines Workflows aus dem Modellierungswerkzeug kennen sich die beiden Werkzeuge.

In der nachfolgenden Abbildung 5.7 wird die Architektur des neuen Konzepts dargestellt. Die neue Architektur für die visuelle Korrelation ordnet sich der Prozessverwaltungskomponente aus der Architektur von [Eic10] unter. Die Engine versendet Events an das Modellierungswerkzeug und an das Auditing Tool. Das Auditing Tool speichert die Events in der Datenbank ab. Im Modellierungswerkzeug bilden die XPathMap, der Monitoring Provider und der Event Model Provider die Datenbasis. Aus ihr beziehen die überliegenden Schichten ihre Daten. Die Events werden im Modellierungswerkzeug von den spezifischen Handlern bearbeitet und an die dementsprechenden Manager weitergeleitet. Über den XPathMapper können die Manager auf das Prozessmodell zugreifen und Änderungen, wie Zustandsänderungen einer Aktivität anzeigen oder den Wert einer Variablen ändern, durchführen. Nach der Bearbeitung erhält die Auditing View die Information und zeigt die Audit Informationen an, sodass der Wissenschaftler alles genau verfolgen kann. Im Auditing Tool werden die Events von der Engine entgegengenommen, persistiert und angezeigt. Dabei verhält es sich ähnlich

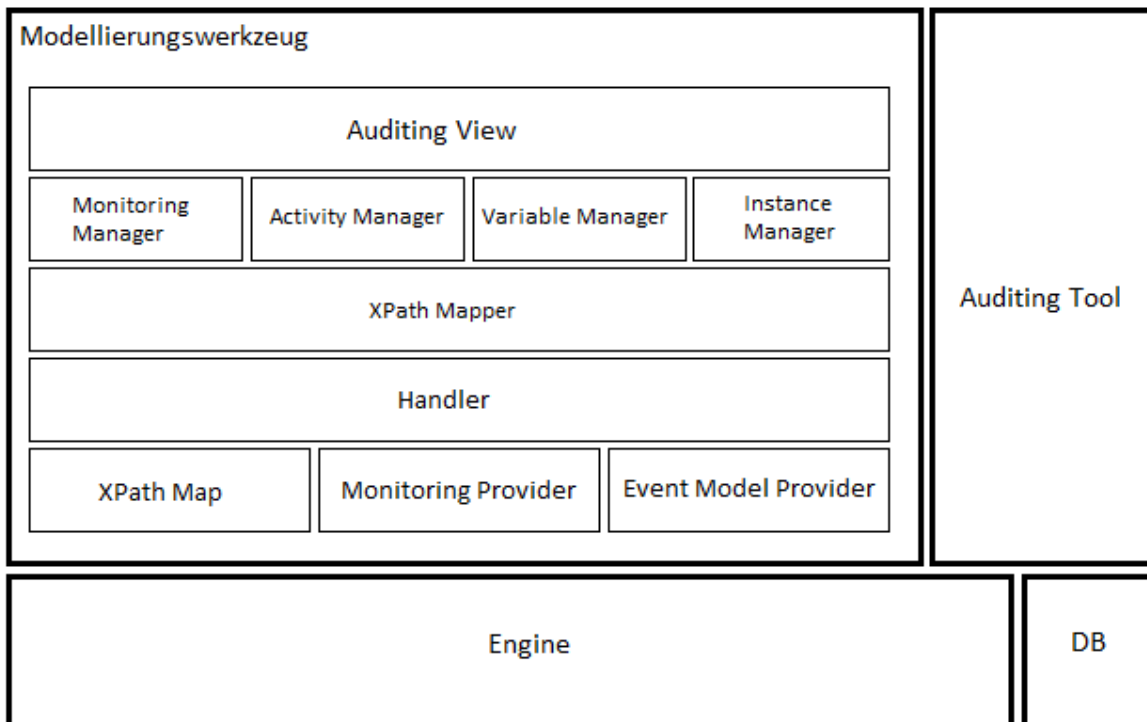


Abbildung 5.7.: Architektur des neuen Konzepts zur visuellen Korrelation

wie das Modellierungswerkzeug. Das Auditing Tool greift nicht auf das Prozessmodell zu, da es nur Informationen bezüglich der Events anzeigt und keinen Prozess.

6. Implementierung

Nachdem das Konzept nun vorgestellt wurde, muss dieses auch realisiert werden. In diesem Kapitel soll nun das zuvor beschriebene Konzept verwirklicht werden. Es wird auf die Feinheiten der Implementierung eingegangen und gezeigt, wie das Konzept im Zuge dieser Arbeit implementiert wird. Das Konzept kann natürlich auch auf andere Art und Weise verwirklicht werden. Die nachfolgende Implementierung ist nur eine Möglichkeit.

6.1. Parametrisiertes Starten von Workflows

Für das parametrisierte Starten der Workflows spielen die vorgestellten Komponenten im Kapitel 5 eine wichtige Rolle. Nachfolgend wird die Implementierung der einzelnen Komponenten erläutert.

6.1.1. Parser

Der Parser ist dafür zuständig, die zuvor markierten Parametervariablen zu identifizieren und in eine Liste abzuspeichern. Zunächst muss aber das BPEL Schema des Eclipse BPEL Designers erweitert werden. Wie in Kapitel 2 bereits erwähnt, arbeitet der Eclipse BPEL Designer mit dem Eclipse Modeling Framework. Dieses stellt das Modell für die BPEL Spezifikation bereit. Dieses gilt es nun zu erweitern. Im Paket *org.eclipse.bpel.model.model* befindet sich die **bpel.ecore** und die **bpel.genmodel** Datei. Dabei ist **bpel.ecore** die Modelldatei, die das BPEL Schema repräsentiert. Hier wird dem *Variable* Element ein zusätzliches Attribut (EAttribute) als Kindknoten hinzugefügt. Das Attribut ist vom Typ EBoolean, welches einen Wahrheitswert repräsentiert und somit speichert, ob eine Variable als Parametervariable markiert ist oder nicht. Die Abbildung 6.1 zeigt die Erweiterung des BPEL Schemas in der **bpel.ecore** Datei.

Nachdem das Schema erweitert wurde, muss nur noch das Modell generiert werden. Das kann mit Hilfe der **bpel.genmodel** Datei erreicht werden. Aus dem EMF-Modell werden alle nötigen Modellklassen generiert. Es wird jeweils ein Interface und eine Implementierung des Interfaces generiert. Für das Setzen des neuen Attributs wird der unten stehende Code 6.1 in der Klasse *VariableImpl* aufgerufen.

Damit die Variablen als Parametervariablen markiert werden können, wird in der grafischen Oberfläche des Eclipse BPEL Designers eine Check Box hinzugefügt, die eine ausgewählte

6. Implementierung

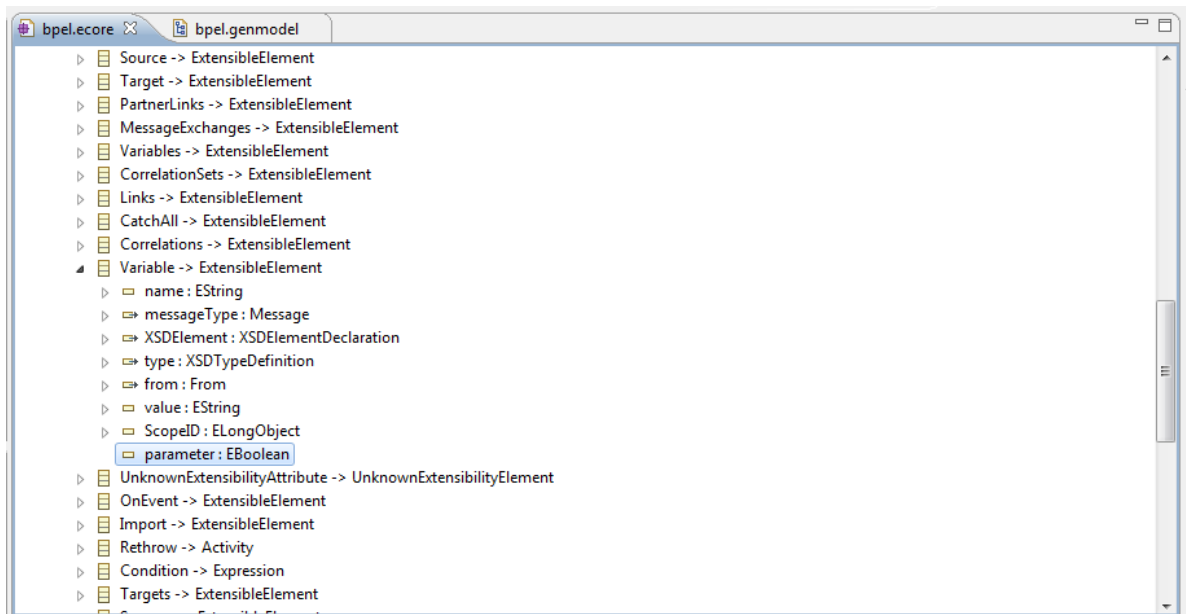


Abbildung 6.1.: Erweiterung des BPEL Schemas

Listing 6.1 Setzen des **parameter** Attributs

```
public void setParameter(boolean newParameter) {
    boolean oldParameter = parameter;
    if (!isReconciling) {
        ReconciliationHelper.replaceAttribute(this, BPELConstants.AT_PARAMETER,
            BPELUtils.boolean2XML(newParameter));
    }

    parameter = newParameter;
    if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
            BPELPackage.VARIABLE__PARAMETER, oldParameter,
            parameter));
}
```

Variable als Parametervariable markiert. In Abbildung 6.2 ist die Check Box im Eclipse BPEL Designer zu sehen, die es ermöglicht Variablen als Parametervariablen zu markieren.

Die Check Box wird in der Klasse *VariableInitializationSection* des Paketes *org.eclipse.bpel.ui.properties* angelegt. Bei Auswahl der Check Box wird das entsprechende Modell verändert. Wird die Check Box angeklickt, so wird durch das Command Pattern der *SetParameterCommand*, der im Zuge dieser Arbeit erstellt wurde, ausgeführt. Der *SetParameterCommand* erwartet eine ausgewählte Variable und den Wert der Check Box. Danach wird im Variable Modell das Attribut **parameter** je nach Wert der Check Box auf true oder false gesetzt. Somit werden die Änderungen an der grafischen Oberfläche direkt

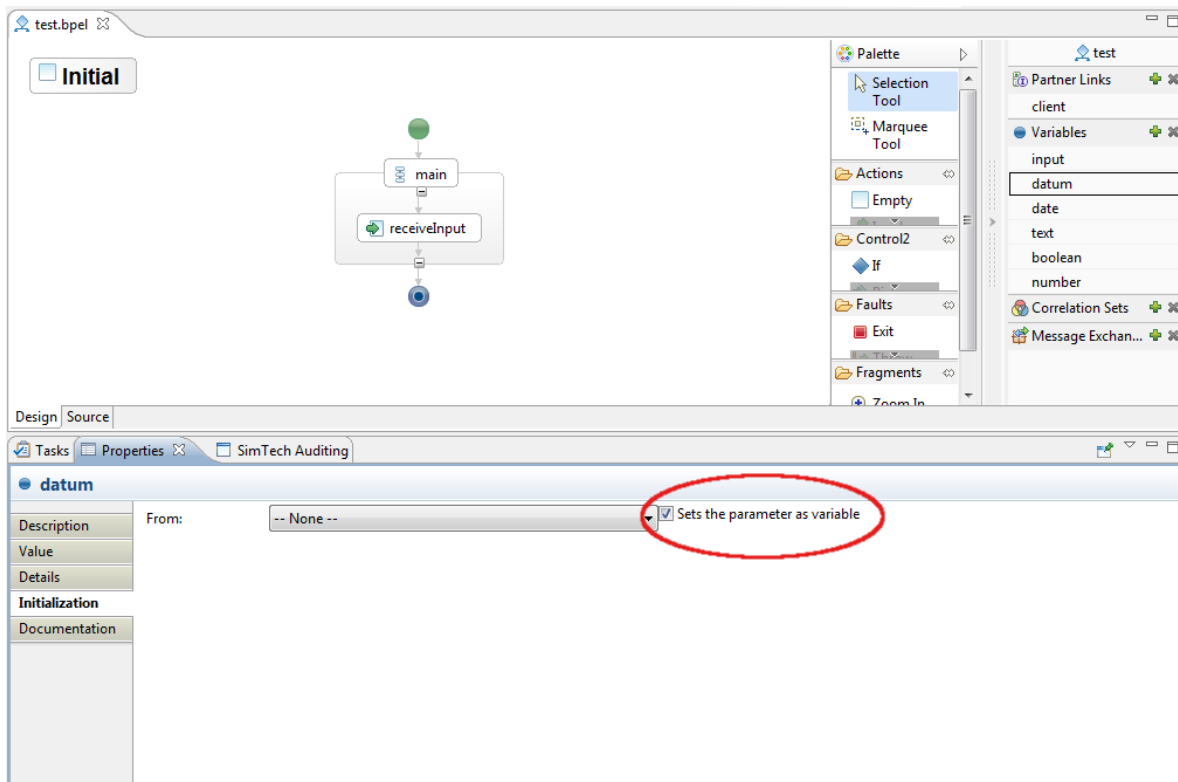


Abbildung 6.2.: CheckBox zur Markierung von Parametervariablen

auf das Modell übertragen. Zusätzlich wird die markierte Variable in die WSDL Datei über den Parameter Handler eingetragen. Der Code in Listing 6.2 erledigt diese Aufgaben.

Nachdem nun alle Variablen markiert sind, muss nur noch der Prozess geparkt werden, um die Variablen heraus zu filtern, die als Parametervariablen gekennzeichnet sind. Das Parsen übernimmt die im Zuge dieser Arbeit neu angelegte Klasse *VariableParser* im Paket *org.eclipse.bpel.ui.parameters*. Der Parser greift über den *BPELMultipageEditor* auf den geraden aktiven Prozess zu. Der Prozess hält alle Variablen in einer Liste. Über diese Liste der Variablen wird nun iteriert und alle Variablen heraus gefiltert, die im Attribut **parameter** den Wert yes haben. Das Konzept wird eingeschränkt, indem nur die Variablen des äußersten Scopes, dem Prozess-Scope, geparkt werden. Danach wird über den Parameter Handler eine Komponente des Parameter Dialogs, anhand der Variable, aufgebaut. Letztendlich wird diese Variable in die HashMap im Parameter Handler abgespeichert. Um zu kennzeichnen, dass sich in einem Prozessmodell keine Parametervariablen befinden, wird der *noParameter* Flag auf true gesetzt. Der Flag dient dazu, dem Parameter Handler zu signalisieren, dass es Parametervariablen im Prozess gibt und der Dialog aufgebaut und geöffnet werden muss. Es ist durchaus möglich, dass keine Parametervariablen existieren, so kann dann der Workflow sofort gestartet werden und es wird kein Dialog geöffnet. Der nachfolgende Code 6.3 parst den Prozess und baut aus den identifizierten Parametervariablen den Dialog.

Listing 6.2 Setzen der Check Box und Änderung des Variable Modells

```
private void createParameterCheckBox(Composite composite, FlatFormData data) {
    fParameter = fWidgetFactory.createButton(composite,
        Messages.SetVariableParameter, SWT.CHECK);
    data = new FlatFormData();
    fParameter.setLayoutData(data);

    fParameterController = createEditController();
    fParameterController.setFeature(BPELPackage.eINSTANCE.getVariable_Parameter());
    fParameterController.setViewIValue(new ButtonIValue(fParameter));
    fParameterController.startListeningTo(fParameter);

    fParameter.setEnabled(false);
    fParameter.addSelectionListener(new SelectionListener() {

        @Override
        public void widgetSelected(SelectionEvent e) {
            getBPELEditor().getCommandFramework().execute(new
                SetParameterCommand(fVariable, fParameter.getSelection()));
            if (fParameter.getSelection()) {
                ParameterHandler.getInstance().addElement(fVariable);
            } else {
                ParameterHandler.getInstance().deleteElement(fVariable);
            }
        }

        @Override
        public void widgetDefaultSelected(SelectionEvent e) {
            widgetSelected(e);
        }

    });
}
```

6.1.2. Parameterdialog

Wie bereits in Kapitel 5 beschrieben, wird der Parameterdialog mit Hilfe des Composite Pattern realisiert. Dabei beschreibt das Composite Pattern, dass Komponenten in Baumstrukturen zusammengefügt werden. Somit ist es möglich Teil-Ganzes-Hierarchien zu repräsentieren [GHJ09]. Durch eine abstrakte Klasse werden sowohl Blätter als auch ihre Behälter (Äste) repräsentiert. Der Dialog selber beinhaltet einen großen Behälter, der mehrere kleinere Behälter enthalten kann. Diese kleineren Behälter sind dabei Blätter und werden nach den Datentypen aufgeteilt. Dadurch ist es relativ einfach für neue Datentypen den Dialog zu erweitern bzw. zu reduzieren. Im Zuge dieser Arbeit wird die abstrakte Klasse *DialogEntryComponent* erstellt. Die Klasse *DialogEntryComponent* erweitert zudem die Klasse *Composite*. Diese SWT-Klasse kann weitere Widgets halten und ist somit der äußere Container des Dialogs. Die Klasse *DialogEntryComponent* bietet Methoden zur Verwaltung weiterer Komponenten an. Jede Komponente erbt von dieser Klasse. Je nachdem, ob die Komponente weitere Komponenten halten kann oder ob sie eine Blattkomponente ist, werden die jewei-

Listing 6.3 Parsen des Prozesses nach Parametervariablen und Aufbau der Komponenten im Dialog

```
public void parse() {
    Process process = editor.getProcess();
    Variables variables = process.getVariables();
    EList<Variable> variableList = variables.getChildren();
    for (Variable variable : variableList) {
        if (variable.isParameter()) {
            //Build the dialog depending on what type the variable is
            ParameterHandler.getInstance().buildDialog(variable);
            //A parameter variable was found so set the noParameter
            attribute.
            //The system knows that there are parameter variables and the
            parameter
            //dialog will be opened
            noParameter = false;
            if
                (!ParameterHandler.getInstance().getParameters().containsKey(variable))
                    ParameterHandler.getInstance().getParameters().put(variable,
                        null);
        }
    }
    ParameterHandler.getInstance().setNoParameter(noParameter);
}
```

Listing 6.4 Iteration über die Liste und Aufruf der **buildEntry()** Methode

```
@Override
public void buildEntry(String variableName) {
    Iterator<DialogEntryComponent> iterator = entries.iterator();
    while (iterator.hasNext()) {
        DialogEntryComponent component = (DialogEntryComponent) iterator.next();
        if (component.getVariable().getName().equals(variableName)) {
            component.buildEntry(variableName);
        }
    }
}
```

ligen Methoden überschrieben. So ist die Klasse *Entries* die Oberklasse, die alle weiteren Komponenten in eine Liste abspeichert. Ausgehend von dieser Oberklasse *Entries* werden alle Komponenten aufgebaut, indem in jeder Komponente die Methode **buildEntry()** aufgerufen wird. Dadurch entsteht eine Baumstruktur. Dafür wird über die Liste mit den gespeicherten Komponenten iteriert und die Methode aufgerufen. Der folgende Code 6.4 stellt die Iteration über die Liste und das Aufrufen der jeweiligen **buildEntry()** Methode dar.

Für diese Arbeit wird das Konzept des Parameterdialogs dahingehend eingeschränkt, dass im Parameterdialog nur vier Datentypen angezeigt werden. Die vier Datentypen sind: String, Integer, Boolean und Date. Für sie gibt es dementsprechend vier Klassen, welche die jeweiligen Einträge für die Datentypen darstellen. Diese Klassen sind *StringEntry*, *NumberEntry*, *DateTimeEntry* und *BooleanEntry*. Diese Entry Klassen werden im Zuge dieser Arbeit

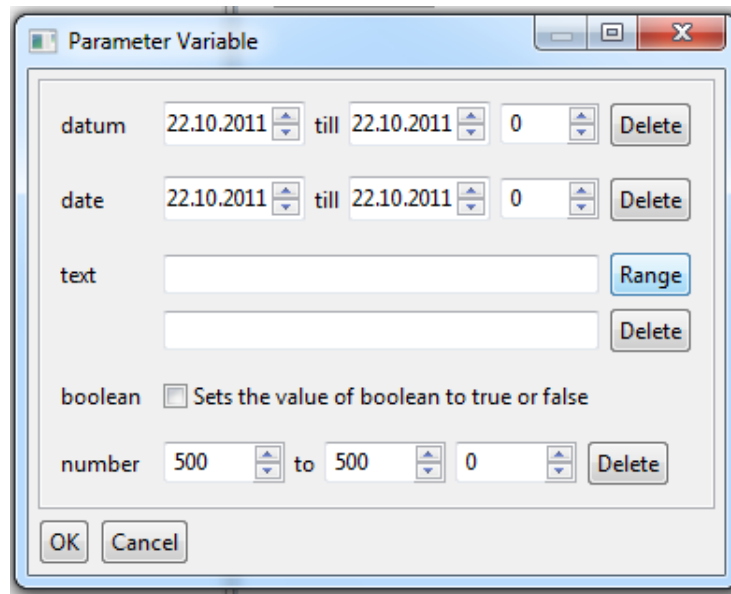


Abbildung 6.3.: Parameterdialog

erstellt. Beim Datentyp Integer werden noch die Datentypen short, decimal und double miteinbezogen. Jeder Entry hält dabei auch die ausgewählte Variable, somit ist jede grafische Komponente mit seinem Modell verbunden und kann daraus Informationen beziehen. Ebenfalls wird je nach Entry die Methode **getValues()** unterschiedlich implementiert. Dabei wird aber immer ein String-Array zurückgeliefert. Somit können nicht nur einzelne Werte, sondern auch mehrere Eingaben gespeichert werden. Falls in Zukunft weitere Datentypen hinzugefügt werden sollen, muss nur die Klasse *DialogEntryComponent* erweitert und die entsprechenden Methoden überschrieben werden. Anschließend muss im Parameter Handler in der Methode **buildDialog()** der entsprechende Entry in die Liste eingefügt werden. Eine genaue Erläuterung folgt im nächsten Abschnitt. Alle Entries und der Dialog befinden sich im Paket *org.eclipse.bpel.parameters.ui*. Die Abbildung 6.3 zeigt den Parameterdialog mit einigen Einträgen.

6.1.3. Parameter Handler

Der Parameter Handler ist die Schaltzentrale zwischen dem Parser und dem Dialog. Er verwaltet alle identifizierten Parametervariablen und speichert sie in eine HashMap mit ihrem Wert, den der Wissenschaftler spezifizieren muss. Der Parameter Handler ist als Singleton Instanz implementiert. Zusätzlich hält der Parameter Handler die Entry Oberklasse, mit deren Hilfe alle weitere Komponenten aufgebaut werden. Nachdem beim Parsen die Parametervariablen identifiziert werden und anhand ihres Datentyps der Eintrag aufgebaut wird, bietet der Parameter Handler die Methode **buildDialog(Variable variable)** an, die alle neuen Komponenten in die Entry Oberklasse abspeichert. Sollte für einen Datentypen noch

Listing 6.5 Hinzufügen der jeweiligen Komponenten in die Entry Oberklasse

```
public void buildDialog(Variable variable) {
    Composite parent = getComponent();
    if (variable.getType().getName().equalsIgnoreCase("string")) {
        getComponent().add(new StringEntry(parent, SWT.NONE, variable));
    } else if (variable.getType().getName().equalsIgnoreCase("boolean")) {
        getComponent().add(new BooleanEntry(parent, SWT.NONE, variable));
    } else if (variable.getType().getName().equalsIgnoreCase("datetime")) {
        getComponent().add(new DateTimeEntry(parent, SWT.NONE, variable));
    } else if (variable.getType().getName().equalsIgnoreCase("int") ||
        variable.getType().getName().equalsIgnoreCase("decimal") ||
        variable.getType().getName().equalsIgnoreCase("double") ||
        variable.getType().getName().equalsIgnoreCase("short")) {
        getComponent().add(new NumberEntry(parent, SWT.NONE, variable));
    } else {
        System.out.println("For this parameter type there isn't an entry
            implemented yet");
    }
}
```

Listing 6.6 Initialisierung des Parameterdialogs

```
public boolean initDialog() {

    Shell parent = PlatformUI.getWorkbench().getDisplay().getActiveShell();
    dialog = new ParameterDialog(parent, SWT.NULL);
    dialog.buildShell();

    component = new Entries(dialog.getDialogShell(), SWT.BORDER);

    parser.parse();

    if (isNoParameter()) {
        return true;
    } else {
        for (Variable variable : getParameters().keySet()) {
            component.buildEntry(variable.getName());
        }
        return false;
    }
}
```

keine Realisierung existieren wird das mit einer Fehlermeldung angezeigt. In Listing 6.5 werden die einzelnen Komponenten der Entry Oberklasse hinzugefügt, je nach Datentyp der Variable.

Der Code 6.6 zeigt die Initialisierung des Dialogs. Der Dialog wird an dieser Stelle noch nicht geöffnet, sondern nur mit den entsprechenden Komponenten initialisiert.

Zunächst wird die Entry Oberklasse in eine Variable gespeichert und der Parser aufgerufen. Nachdem dann die Paramtervariablen identifiziert wurden, wird über die gespeicherten Komponenten iteriert und die jeweilige **buildEntry()** Methode der Komponente aufgerufen.

Listing 6.7 Kartesisches Produkt^a

```
public static List<List<String>> cartProduct(List<List<String>> sets) {
    List<List<String>> cartesian_product=new ArrayList<List<String>>();
    List<String> cartesian_product_element;
    int n=1;
    Iterator<?> sets_it=sets.iterator();
    //loop to get cardinality of Cartesian product
    while (sets_it.hasNext()) {
        List<?> set = (List<?>) sets_it.next();
        n*=set.size();
    }
    //loop to create all elements of Cartesian product
    for (int i=0;i<n;i++){
        int j=1;
        cartesian_product_element=new ArrayList<String>();
        sets_it=sets.iterator();
        //loop that collects one element of each class
        while (sets_it.hasNext()) {
            List<?> set = (List<?>) sets_it.next();
            cartesian_product_element.add(String.valueOf(set.get((i/j)\%set.size())));
            j*=set.size();
        }
        cartesian_product.add(cartesian_product_element);
    }
    return cartesian_product;
}
```

^a<http://www.java-forum.org/413543-post10.html>

Falls keine Parametervariablen vorhanden sind, wird der Dialog nicht geöffnet. Dies geschieht in der *StartAction*, da der Benutzer auf das Start Symbol klicken muss, um einen Workflow zu starten. Dort wird dann, wenn nötig, die **open()** Methode vom Parameterdialog aufgerufen, welche den Dialog öffnet und die Parametervariablen anzeigt. Ansonsten wird der Workflow direkt über den Monitor Manager gestartet. Der Parameter Handler befindet sich im Paket *org.eclipse.bpel.ui.parameters.handler*.

6.1.4. Nachrichtenaufbau mit Hilfe des kartesischen Produkts

Da Wissenschaftler entweder einen einzelnen Wert für einen Parameter oder einen ganzen Wertebereich angeben können und somit unterschiedliche Workflows starten, müssen für alle diese Werte Nachrichten aufgebaut werden, die an die Apache ODE gesendet werden. Wie bereits vorgestellt, wird das mit Hilfe des kartesischen Produkts gelöst, welches aus einer Menge an Tupeln alle Möglichkeiten berechnet. Nun gilt es, das für die Nachrichten zu realisieren. Listing 6.7 zeigt den Code, der das kartesische Produkt realisiert. Der Parameter Handler bietet die Methode zur Berechnung des kartesischen Produktes an.

Die Methode erwartet eine Liste. Ein Listeneintrag ist eine weitere Liste, die Parameterwerte einer Variable enthält, als String. Es wird über die Liste, die alle Parameterwertlisten enthält,

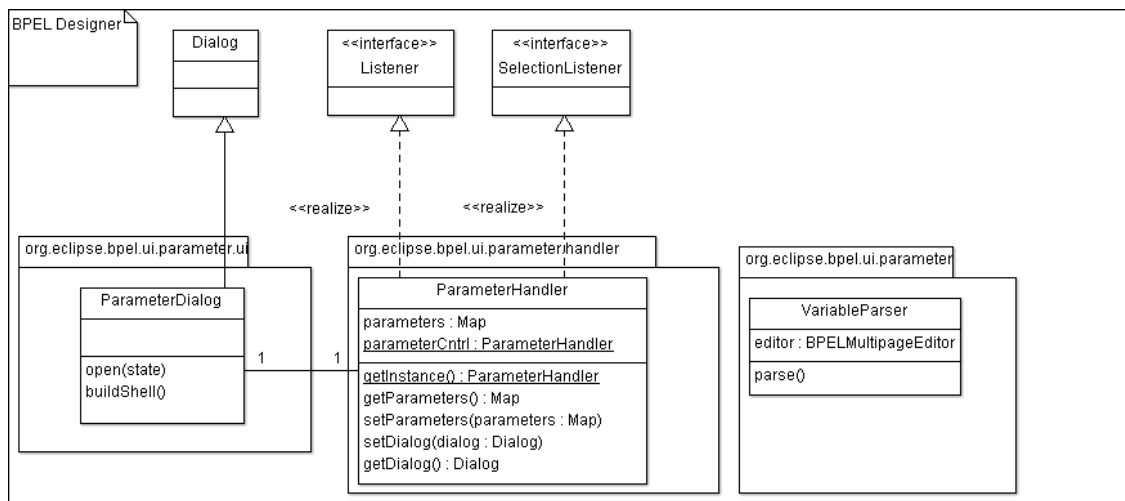


Abbildung 6.4.: Klassendiagramm des parametrisierten Aufrufs

iteriert, um die Kardinalität jeder Parameterwertliste abzufragen und daraus die Kardinalität des kartesischen Produkts zu berechnen. Anschließend wird aus jeder einzelnen Liste der äußeren Liste, mit Hilfe einer Schleife jeweils das erste Element gewählt und in die Ausgabeliste abgelegt. Durch die Iteration steigt der Index immer um eins und durch die Modulo Berechnung werden somit alle Werte ausgewählt, sodass am Ende in der Ausgabeliste alle Tupel des kartesischen Produkts abgelegt sind.

Die Abbildung 6.4 zeigt das Klassendiagramm mit dem Parameterdialog, dem Parameter-Handler und dem Parser. Die komplette Implementierung dieses Konzepts für den Eclipse BPEL Designer wird im Paket *org.eclipse.bpel.ui.parameters* abgelegt. Zusätzliche Änderungen, die mit dem Konzept einhergehen, werden in den entsprechenden Klassen vorgenommen. Das BPEL Schema wird erweitert und somit auch das EMF-Modell des BPEL Designers. Für die Realisierung auf der Seite der Apache ODE wird nur die Pick-Aktivität erweitert, sodass das Befüllen der Parametervariablen auf der Engine Seite durchgeführt werden kann.

6.1.5. Apache ODE

Auf Seiten der Apache ODE muss eine Klasse verändert bzw. erweitert werden, damit sie die Nachricht zum Starten eines wissenschaftlichen Workflows korrekt bearbeiten kann. Ziel ist es nun, die Informationen, also die Parametervariablen mit den dazugehörigen Werten, auszulesen und korrekt in den Prozess einzuordnen. Wie im Kapitel 5 bereits erwähnt, soll das beim Receive/Pick verwirklicht werden. Die Apache ODE unterscheidet nicht wie in der BPEL Spezifikation zwischen Pick und Receive. Pick und Receive werden hier gleich behandelt, daher muss die Änderung in der Klasse *PICK* erfolgen. Da zur Laufzeit erst die Variablen vorhanden sind, muss die Änderung auf Instanzebene durchgeführt werden. Daher

Listing 6.8 Befüllen der Parametervariablen

```
private void fillParameters(String mexId, OnMessage onMessage) throws
    TransformerException, ExternalVariableModuleException {
    Element element = getBpelRuntimeContext().getMyRequest(mexId);

    Element parameter = DOMUtils.findChildByName(element, new QName("parameters"));

    NodeList parameterList = parameter.getFirstChild().getChildNodes();
    for (int i = 0; i < parameterList.getLength(); i++) {
        Element parameterNode = (Element)parameterList.item(i);
        String variableName = parameterNode.getLocalName();

        extractNode(parameterNode, variableName);
    }
}
```

befindet sich die Klasse *PICK* im Paket *org.apache.ode.bpel.runtime*. Die Klasse wird im Zuge dieser Arbeit erweitert. Hier greift das Jacob-Framework zu und verwaltet die Aktivitäten zur Laufzeit. Wie schon bereits erwähnt, müssen die Informationen aus der Nachricht extrahiert und in die richtige Variable im Prozess eingeordnet werden. Das geschieht in der Methode **fillParameter(String mexID, OnMessage onMessage)**. Der Code 6.8 repräsentiert die *fillParameter* Methode.

Das Element der Nachricht wird über die *mexID* (Message Exchange ID) extrahiert. Daraufhin wird über die Kinder des **parameter** Kindes in der Nachricht iteriert, welches die Parametervariablen und ihre Werte enthält. Die Variablen der Nachricht werden mit den Variablen des Prozesses verglichen, um zu entscheiden, um welche Variablen es sich handelt. Dabei werden nur die Variablen des Prozess-Scope berücksichtigt. Anschließend wird der neue Wert aus der Nachricht gelesen und in die Variable gespeichert. Für den Vergleich wird der Variablenname herangezogen, da dieser im Scope eindeutig ist. Listing 6.9 zeigt, wie der gewünschte Knoten aus der Nachricht extrahiert und in die Variable gespeichert wird.

Variablenwerte in der Apache ODE müssen als DOM-Baum abgespeichert werden. Daher muss zunächst der Knoten in die geeignete Form gebracht werden, da auch abhängig vom Typ, der DOM-Baum unterschiedlich aussieht. So muss für einen simplen Datentypen ein **temporary-simple-type-wrapper** als Element Namespace herangezogen werden. Bei anderen Datentypen kann der Wert des Knoten einfach an den DOM-Baum angehängt werden. Der Code 6.10 stellt den neu aufbereiteten Knoten als DOM-Baum dar.

6.2. Visuelle Korrelation von parallelen Workflows

Im Eclipse BPEL Designer soll die visuelle Korrelation von parallelen Workflows realisiert werden. Nachdem in Kapitel 5 das Konzept vorgestellt wurde, wird im nun folgenden Abschnitt die Realisierung vorgestellt.

Listing 6.9 Extraktion und Speicherung des korrekten Knotens

```
private void extractNode(Element parameterNode, String variableName) throws
    TransformerException, ExternalVariableModuleException {

    String scopeName =
        getBpelRuntimeContext().getProcessInstanceDao().getRootScope().getName();
    OScope oScope =
        getBpelRuntimeContext().getBpelProcess().getOProcess().getScope(scopeName);

    for (String varName : oScope.getVariables().keySet()) {
        Variable variable = oScope.getLocalVariable(varName);
        if (!variableName.equals("input") &&
            variable.name.equals(variableName)) {
            Node valueNode = buildDocument(variable, parameterNode);
            VariableInstance varInst = _scopeFrame.resolve(variable);
            initializeVariable(varInst, valueNode);
        }
    }
}
```

Listing 6.10 Bauen eines DOM-Objekts aus dem extrahierten Knoten

```
private Node buildDocument(Variable variable, Element parameterNode) throws
    TransformerException {
    Document doc = DOMUtils.newDocument();
    Node val = variable.type.newInstance(doc);
    if (val.getNodeType() == Node.TEXT_NODE) {
        Element tempwrapper = doc.createElementNS(null,
            "temporary-simple-type-wrapper");
        doc.appendChild(tempwrapper);
        val.setNodeValue(parameterNode.getTextContent());
        tempwrapper.appendChild(val);
    } else {
        val.setNodeValue(parameterNode.getTextContent());
        doc.appendChild(val);
    }

    return val;
}
```

6.2.1. Monitoring Provider

Im Zuge dieser Arbeit soll die parallele Überwachung mehrerer Workflows von mehreren Workflow Modellen ermöglicht werden. Dazu wurde der Monitoring Provider entwickelt, der alle gestarteten Instanzen zur Verfügung stellt. In der Ausgangssituation hielt jeder Manager seine eigene Workflow Instanz. Es konnten zwar auch mehrere Workflow Instanzen gestartet werden, aber nur eine konnte überwacht werden. Nun können die Manager, die eine bestimmte Instanz benötigen, anhand der InstanzID am Monitoring Provider Informationen über die korrekte Instanz anfragen. Listing 6.11 zeigt die Anfrage.

Listing 6.11 Finden einer Instanz anhand ihrer ID

```
public InstanceInformation getInstanceByID(Long instanceID) {
    InstanceInformation instanceInfo = null;
    for (InstanceInformation instance : correlationMap.keySet()) {
        if (instance.getInstanceID() != null &&
            instance.getInstanceID().equals(instanceID)) {
            instanceInfo = instance;
        }
    }
    return instanceInfo;
}
```

Der Monitoring Provider ist als Singleton implementiert, sodass nur eine Instanz des Providers vorhanden ist und immer auf der gleichen HashMap gearbeitet wird. Der Monitoring Provider speichert zum einen alle Instanzen mit ihren dazugehörigen Monitor Managern in eine HashMap ab und zusätzlich speichert der Monitoring Provider zu jedem Prozess, der aktuell im BPEL Designer angezeigt wird, die aktuell im Monitor angezeigte Instanz. Dadurch kann über den im Designer angezeigten Prozess direkt auf die aktuelle Instanz zugegriffen werden und es ist keine InstanzID nötig. Dies ist notwendig, da zum Beispiel beim Stoppen oder Pausieren eines Prozesses, im Eclipse BPEL Designer, die InstanzID nicht vorhanden ist, jedoch aber der Prozess, der gerade angezeigt wird.

6.2.2. Monitor Manager

Damit parallele Workflows überwacht werden können, muss zunächst aus der Singleton Instanz des Monitor Manager eine normale Klasse erstellt werden, die es ermöglicht, mehrere Instanzen des Monitor Managers zu erzeugen. Der neue Monitor Manager hält die Instanz, die er überwacht. Die Klasse *InstanceInformation* repräsentiert eine Instanz. Die Methoden, die der Monitor Manager für Instanzen zur Verfügung stellt, wie **start()**, **stop()**, **resume()**, **restart()** und **suspend()** müssen an die neuen Gegebenheiten angepasst werden. Im Ausgangszustand wurde jeweils die zuletzt gestartete Instanz verändert, das heißt, beim Aufruf von **suspend()** wurde die letzte gestartete Instanz pausiert. Da durch die Erweiterung nun alle Instanzen durch den Monitoring Provider zur Verfügung gestellt werden, muss zunächst die korrekte Instanz gewählt werden, um die Informationen im BPEL Designer korrekt anzeigen zu können. Die korrekte Instanz eines Prozessmodells ist die zuletzt über den Eclipse BPEL Designer gestartete Instanz des Modells. Der Wissenschaftler hat auch die Möglichkeit durch einen bereits existierenden Dialog andere laufende Instanzen für das Monitoring auszuwählen. Die ganzen Methoden wirken sich dann auf die ausgewählte Instanz aus. Beim Monitor Manager sind keine neuen Methoden hinzugefügt worden, lediglich die bestehenden wurden angepasst, sodass sie mit mehreren Instanzen umgehen können. Dies hat zur Folge, dass weitere Manager ebenfalls angepasst werden müssen, damit sie den Bedingungen entsprechen und mit mehreren Instanzen umgehen können. Die Abbildung 6.5 zeigt einen Ausschnitt aus dem Klassendiagramm, wie der Monitor Manager, der Monitoring Provider und die Instance Information zusammen hängen.

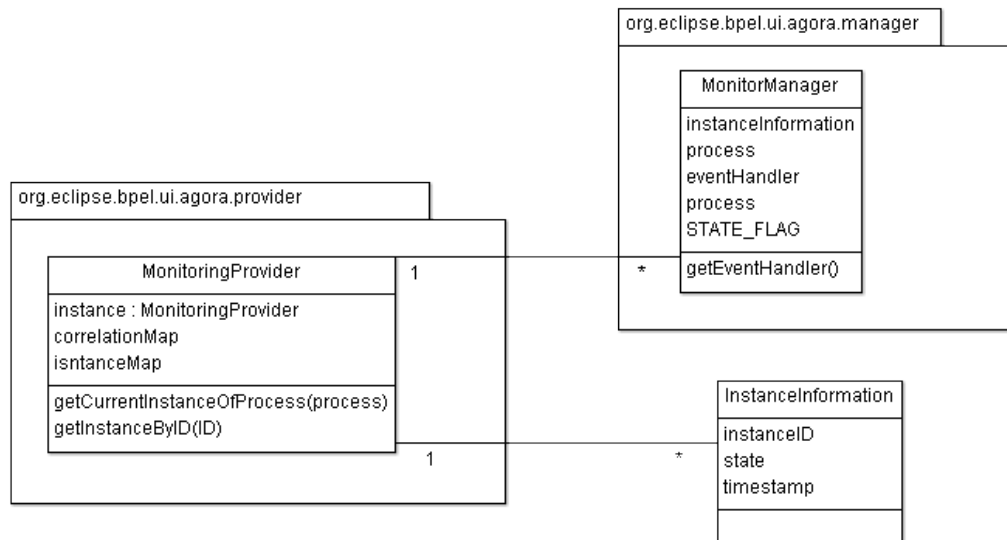


Abbildung 6.5.: Klassendiagramm Monitor Manager, Monitor Provider und Instance Information

6.2.3. Instance Manager

Der Instance Manager hat im Zuge der Realisierung des Konzepts zur visuellen Korrelation viele seiner Methoden verloren. Da die Verwaltung der Instanzen nun der Monitoring Provider übernimmt, muss sich der Instance Manager nicht mehr um die Instanz kümmern. Nun wird zunächst am Monitoring Provider nach der richtigen Instanz angefragt. Dies geschieht mit Hilfe der InstanzID, die in der Nachricht enthalten ist, die gesendet wird, wenn ein bestimmtes Event auftritt. Daraufhin kann der Instance Manager mit der korrekten Instanz weiter arbeiten. Neu hinzugefügt wurde die Methode **handleProcessUndeployed**. Diese Methode setzt im Falle des Undeployment eines Prozesses die Sicht des BPEL Designers und das Prozessmodell auf den Ursprungszustand zurück und löscht die Instanz aus der HashMap des Monitoring Provider. Somit befindet sich die Sicht im Ausgangszustand und es kann eine neue Instanz gestartet werden. Das im Konzept vorgestellte Auditing, worauf erst später konkret eingegangen wird, hält alle Events bezüglich eines Prozesses. Beim Undeploymen eines Prozesses werden diese Events aus dem EventModelProvider gelöscht. Eine genau Erläuterung befindet sich im Abschnitt 6.2.5. Der Code 6.12 zeigt die Methode **handleProcessUndeployed**.

Der Activity- und Variable Manager werden dahingehend angepasst, dass die bestehenden Methoden einen Monitor Manager als Parameter erhalten, damit aus ihm in der späteren

Listing 6.12 Bearbeitung eines undeployten Prozesses

```
public void handleProcessUndeployed(Process_Undeployed message) {
    for (InstanceInformation instanceInfo :
        MonitoringProvider.getInstance().getCorrelationMap().keySet()) {
        if (instanceInfo.getProcessName().equals(message.getProcessName()) &&
            !instanceInfo.getState().equals(BPELStates.Initial)) {
            MonitorManager manager =
                MonitoringProvider.getInstance().getCorrelationMap().get(instanceInfo);
            instanceState = BPELStates.Initial;
            EventModelProvider.getInstance().clear(manager.getProcess());
            updateView(manager);
            InstanceManager.getInstance().deleteInstance(instanceInfo);
        }
    }
}
```

Bearbeitung der dort gehaltene Prozess ausgelesen und bearbeitet werden kann. Beim Activity Manager werden die einzelnen Aktivitäten bearbeitet, die anhand ihres XPath Ausdrucks identifiziert werden. Der XPath Ausdruck wird über die Nachricht mitgesendet, allerdings muss im Eclipse BPEL Designer das korrekte Prozessmodell zu der jeweiligen Instanz ausgewählt werden. In der Ausgangssituation konnte der Activity Manager nur die zuletzt gestartete Instanz bearbeiten. Im Zuge dieser Arbeit wird dem Activity Manager der Monitor Manager als Parameter übergeben. Somit weiß der Activity Manager genau, welche Instanz er bearbeiten soll, da der Monitor Manager die zu überwachende Instanz speichert und auf welche Instanz sich die Events beziehen, die von der Engine kommen. Das selbe gilt für den Variable Manager. Der erhält den Monitor Manager ebenfalls als Parameter. Die eigentliche Aufgabe übernimmt dann der XPathMapper, dessen Änderungen im nachfolgenden Abschnitt erläutert werden.

6.2.4. XPathMapper und XPathMap

Der XPathMapper ist dafür zuständig, anhand eines XPath Ausdrucks ein Element im EMF-Modell zu lokalisieren und entsprechende Änderungen, wie z.B. Zustandsänderungen, an diesem Element durchzuführen. Dieser fragt bei der Klasse *XPathMap* nach dem Element zum dazugehörigen XPath Ausdruck. Im Zuge dieser Arbeit wird die *XPathMap* Klasse dahingehend verändert, dass sie zur Speicherung der XPath Ausdrücke eine HashMap verwendet, deren Schlüssel der jeweilige Prozess ist und als Werte eine Liste der XPath Ausdrücke speichert. Somit kann unabhängig voneinander zu jedem Prozess der jeweilige XPath Ausdruck ausgewertet und das korrekte Element gewählt werden. Dieses Element wird dann in der Klasse *XPathMapper* verwendet, um zum Beispiel den Zustand einer Aktivität oder den Wert einer Variablen zu verändern.

instanceindex	instanceID
INSTANCEID BIGINT(20)	messageID BIGINT(20)
STATE VARCHAR(255)	event_type VARCHAR(255)
PROCESS_PATH VARCHAR(1024)	object BLOB
PROCESS_NAME VARCHAR(1024)	timing BIGINT(20)
PACKAGE_NAME VARCHAR(1024)	activity_XPath VARCHAR(512)
PROCESS_VERSION BIGINT(20)	state VARCHAR(255)
TIMESTAMP BIGINT(20)	
Indexes	Indexes

Abbildung 6.6.: Datenbanktabellen des Auditing Tool

6.2.5. Auditing

Im Auditing Tool werden die Informationen gespeichert, die zur Überwachung der Instanzen notwendig sind. Dabei wird eine Tabelle erstellt, die alle Instanzen speichert, die gestartet werden. Zusätzlich wird für jede Instanz eine eigene Tabelle angelegt, die weitere Informationen, wie den Zustand einer Aktivität und den XPath Ausdruck speichert. Die Abbildung 6.6 zeigt die Tabellen des Auditing Tools, die angelegt werden und bereits existieren.

Dabei wird der Name der Tabelle instanceID so zusammengesetzt, in dem noch die ID der Instanz angehängt wird und somit die nötigen Informationen abgespeichert werden. Wie die Datenbanktabellen in Abbildung 6.6 zeigen, werden von einer Instanz die ID, der Eventtyp, der Timestamp, der XPath Ausdruck der Aktivität, der Zustand der Aktivität und die Nachricht des Events abgespeichert. Somit sind alle Informationen bezüglich einer Instanz dauerhaft vorhanden. In der instanceindex Tabelle werden alle gestarteten Instanzen abgespeichert. Die instanceindex Tabelle beinhalten die InstanzID, den Zustand der Instanz, der Prozesspfad, der Prozessname, der Paketname, die Prozessversion und der Timestamp. Die gespeicherten Informationen, die im Auditing Tool angezeigt werden, müssen zunächst aus der Datenbank gelesen werden. Das geschieht mit Hilfe der Klasse *MySQLConnector*. Diese Klasse bietet Methoden an, um mit der Datenbank eine Verbindung aufzunehmen bzw. diese zu terminieren und bestimmte Daten aus der Datenbank auszulesen. Im Zuge diese Arbeit soll die Auditing Sicht im Eclipse BPEL Designer dementsprechend angepasst werden, dass die Informationen zur jeweils ausgewählten Instanz angezeigt werden sollen. Dazu müssen zunächst aber die korrekten Informationen aus der Datenbank extrahiert werden.

Listing 6.13 Extraktion der notwendigen Daten für die Auditing Sicht

```
public List<Object[]> getAuditingDataOfInstance(Long instanceID) {
    List<Object[]> auditingList = new ArrayList<Object[]>();

    Connection connection = instance.openConnection();

    if (connection != null) {
        try {
            Statement statement = connection.createStatement();

            ResultSet result = statement
                .executeQuery("SELECT * FROM instance" +
                    instanceID);

            while (result.next()) {
                auditingList.add(new Object[] {
                    result.getString("event_type"),
                    result.getString("activity_XPath"),
                    result.getLong("timing"),
                    result.getString("state"),
                    result.getObject("object") });
            }

            result.close();
            statement.close();

        } catch (SQLException e) {
            e.printStackTrace();
        }

        closeConnection(connection);
    }

    return auditingList;
}
```

Der Listing 6.13 zeigt den Code zur Extraktion der notwendigen Daten für die Auditing Sicht.

Aus der Datenbank werden die Informationen aus der jeweiligen Instanz Tabelle ausgelesen, die notwendig sind, um die Auditing Sicht zu füllen. Diese Informationen sind der Eventtyp, der XPath Ausdruck der Aktivität, der Timestamp, der Zustand der Aktivität und die Nachricht. Die Auditing Sicht kann auf zwei unterschiedliche Arten befüllt werden. Zum einen wird die Auditing Sicht direkt durch die Events, die zu der geraden laufenden Instanz im Monitor gehören, befüllt, zum anderen durch das Öffnen einer beliebigen, persistierten Instanz zum Monitoring. Beim Öffnen einer bereits persistierten Instanz wird eine Anfrage an das Auditing Tool gesendet. Das Auditing Tool liest aus der Anfrage die Instanz ID aus und extrahiert aus der Datenbank die Informationen für die dazu gehörende Instanz. Die Informationen werden dann als Liste in eine Nachricht gespeichert und an den Anfrager zurück gesendet. In der Klasse *RequestEventHandler* wird die Nachricht ausgewertet. Die

Listing 6.14 Anpassung der Sicht im Eclipse BPEL Designer

```

    if (obj instanceof InstanceHistoryMessage) {
        InstanceHistoryMessage historyMessage = (InstanceHistoryMessage) obj;

        HashMap<String, BPELStates> stateHistory = historyMessage
            .getHistory();
        Long instanceID = historyMessage.getProcessInstanceID();

        List<Object[]> objects = historyMessage.getObjects();

        InstanceInformation instanceInfo =
            MonitoringProvider.getInstance().getInstanceByID(instanceID);
        MonitorManager manager =
            MonitoringProvider.getInstance().getCorrelationMap().get(instanceInfo);
        //Set the states of the activities
        if (instanceInfo.getInstanceID().equals(instanceID)){
            for (String xpath : stateHistory.keySet()){
                XPathMapper.setState(xpath, stateHistory.get(xpath),
                    manager);
            }
        }
        //Set the information for the auditing view
        List<EventMessage> events =
            EventModelProvider.getInstance().getEvents(manager.getProcess());

        int i = 0;
        for (Object[] object : objects) {
            EventMessage eventMessage = new EventMessage();
            eventMessage.setEventType((String) object[i]);
            eventMessage.setSource((String) object[i+1]);
            eventMessage.setTimestamp((Long) object[i+2]);
            eventMessage.setState((String) object[i+3] != null ?
                BPELStates.valueOf((String) object[i+3]) : null);
            eventMessage.setMessageObject(object[i+4]);
            events.add(eventMessage);
        }

        EventModelProvider.getInstance().getViewListener().getViewer().setInput(events);
        EventModelProvider.getInstance().updateView();
    }

```

Aktivitäten des Prozesses werden je nach Zustand, der in der Nachricht enthalten ist, markiert. Das ist bereits vorhanden. Im Zuge dieser Arbeit wurde die Auditing Sicht nun mit den korrekten Informationen, dem Eventtyp, dem XPath-Ausdruck usw., die ebenfalls in der Nachricht enthalten sind, befüllt. In Listing 6.14 ist der Code für die Bearbeitung der ankommenden Nachricht, die vom Auditing Tool gesendet wird, aufgeführt.

Die Klasse *EventModelProvider* verwaltet alle Events, die von der Engine gefeuert werden. Allerdings ist die Klasse nur für eine Instanz realisiert worden, sodass Events mehrerer Instanzen nicht gespeichert werden. Im Zuge dieser Arbeit wurde die Klasse dementsprechend

angepasst, dass die Klasse für jeden Prozess die Events speichert. Die Speicherung erfolgt in einer HashMap, die als Schlüssel das Prozessmodell hält und als Wert die gerade gestartete oder ausgewählte Instanz. Im Eclipse BPEL Designer ist es nicht möglich, mehrere Instanzen eines Prozessmodells gleichzeitig anzuzeigen. Somit werden die Events für nur eine Instanz gespeichert. Das bedeutet, dass obwohl von einem Prozessmodell mehrere Instanzen existieren können, werden nicht für alle Instanzen die Events abgespeichert, sondern nur für die Instanz, die gerade gestartet oder ausgewählt wurde. Und da mehrere Instanzen auf ein Prozessmodell mappen und im Eclipse BPEL Designer nur Prozessmodelle angezeigt werden, ist das Prozessmodell der Schlüssel in der HashMap. Somit erhält die Auditing Sicht die Informationen einer Instanz, die gerade ausgewählt bzw. neu gestartet wurde. Durch die Speicherung der Events in der HashMap, wird auch beim Wechsel der einzelnen Prozesstabs im BPEL Designer die Auditing Sicht aktualisiert und jeweils die korrekten Events aus dem EventModelProvider gelesen.

Für die Realisierung waren noch einige weitere kleinere Änderungen an Klassen notwendig, die hier jedoch nicht aufgeführt werden, da sie sich durch die Hauptänderungen ergeben haben. Genaue Informationen, welche Anpassungen gemacht wurden, befinden sich im Code als Kommentar. In diesem Kapitel wurden nur die Hauptänderungen vorgestellt, damit der Leser einen Ansatz hat, um die Änderungen nachvollziehen zu können. Alle Manager, der XPathMapper und der RequestEventHandler befinden sich im Paket *org.eclipse.bpel.ui.agora*. Diese Klassen werden im Zuge der Arbeit erweitert und für das Konzept entsprechend angepasst.

6.3. Anwendungsbeispiel

In diesem Abschnitt soll ein kleines Anwendungsbeispiel dargestellt werden. Der Prozess, der für das Beispiel verwendet wird, stammt aus der Diplomarbeit von [Cheog]. Der Prozess visualisiert das Ergebnis einer Luftströmungssimulation in einem Raum anhand der LIC-Methode (Line Integral Convolution). In Abbildung 6.7 ist der Prozess im Eclipse BPEL Designer dargestellt.

Der Prozess erwartet sechs Parameter, die der Wissenschaftler mit Werten spezifizieren soll. Es handelt sich bei den Parametern um *fieldtype*, *noisetype*, *kerneltype*, *iteration*, *delta* und *integrationsmethod*. Die Parameter *fieldtype*, *noisetype*, *kerneltype* und *integrationsmethod* sind vom Typ String, *iteration* und *delta* sind vom Typ Integer. Die Parameter können folgende Werte annehmen:

fieldtype SynLine, SynWhirl, QuadWhirl, Distort, OSZI, Critpoint, Multi_Critpoint, Random, Sinx, Star

noisetype White, Sparse, Checker, White_G2, Rectangle

kerneltype Gauss, Box, Antisym Gauss

iteration 0 - 99

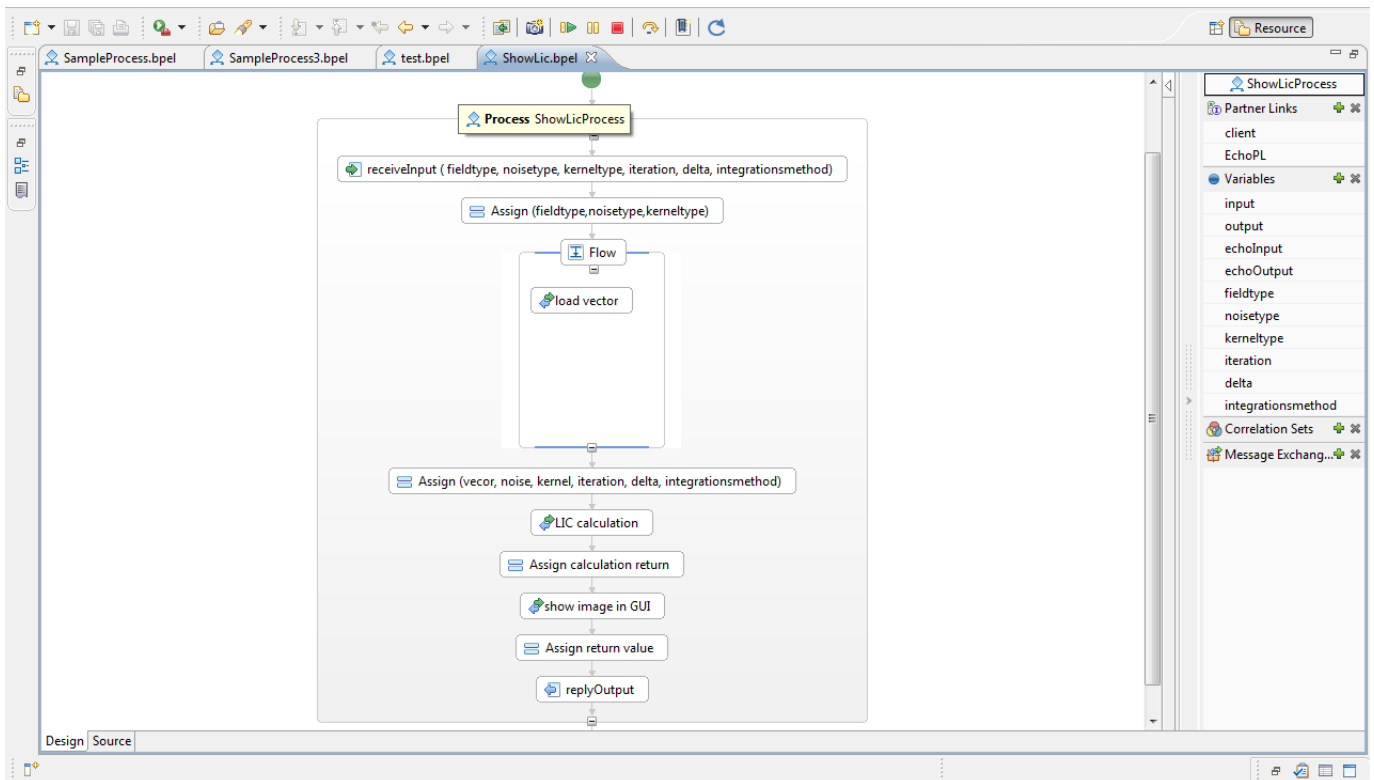


Abbildung 6.7.: Prozess zur Visualisierung von Luftströmen in einem Raum anhand LIC-Method [Cheo9]

delta 0.1 - 0.99

integrationsmethod Euler, RK4

Für alle diese Parameter werden Variablen angelegt, die nun zu Parametervariablen deklariert werden müssen, damit der Wissenschaftler im Dialog die Parameterwerte eintragen kann. Die Abbildung 6.8 zeigt die Stelle im Eclipse BPEL Designer, an dem der Wissenschaftler Variablen zu Parametervariablen deklarieren kann.

Nachdem die Parametervariablen markiert wurden, kann der Wissenschaftler den Prozess starten. Es öffnet sich ein Dialog, in dem er die Werte für die Parametervariablen spezifizieren kann. Dabei besitzt der Wissenschaftler die Möglichkeit einen einzigen Wert oder einen ganzen Wertebereich einzutragen. Die Abbildung 6.9 zeigt den Dialog zur Eingabe der Parameterwerte.

Die Parameterwerte werden in einzelne Nachrichten verpackt, die mit Hilfe des kartesischen Produkts berechnet und an die Engine gesendet werden. Dort wird der Prozess deployt und mit Hilfe der gesendeten Nachricht gestartet. Im Anwendungsbeispiel werden die in Listing 6.15 und Listing 6.16 dargestellten Nachrichten an die Apache ODE gesendet. Die Parametervariable delta kann zwar nur Werte von 0.1 bis 0.99 annehmen, allerdings werden

6. Implementierung

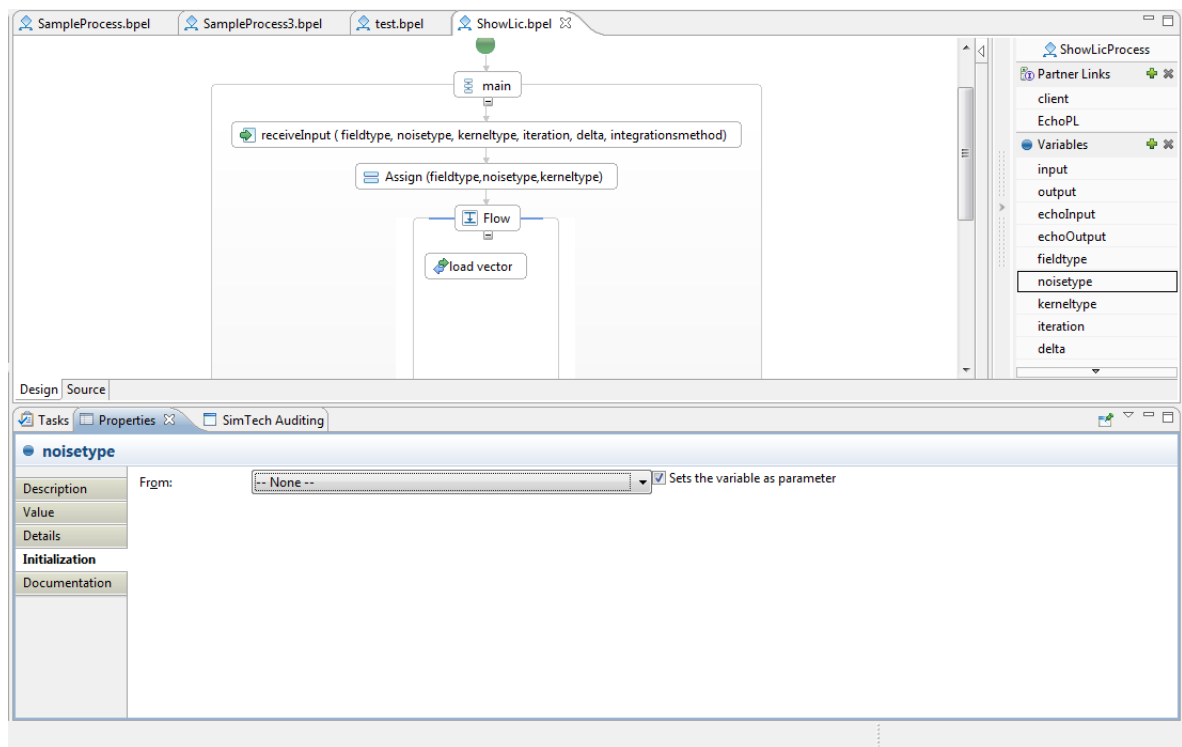


Abbildung 6.8.: Markierung der Variablen Variablen zu Parametervariablen

Listing 6.15 Erste Nachricht, die an die Apache ODE gesendet wird

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wSDL="de.test/wSDL">
  <soapenv:Header/>
  <soapenv:Body>
    <wSDL:testParameterRequest>
      <wSDL:fieldtype>SynLine</wSDL:fieldtype>
      <wSDL:noisetype>White</wSDL:noisetype>
      <wSDL:kerneltype>Gauss</wSDL:kerneltype>
      <wSDL:iteration>20</wSDL:iteration>
      <wSDL:delta>5</wSDL:delta>
      <wSDL:integrationsmethod>Euler</wSDL:integrationsmethod>
    </wSDL:testParameterRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

die Werte für dieses Anwendungsbeispiel mit zehn multipliziert, da der Datentyp float momentan noch nicht unterstützt wird.

Auf der Apache ODE werden die Prozesse deployt. Aus den eingehenden Nachrichten werden die Parameterwerte extrahiert und in die richtigen Variablen im Prozess kopiert. Die Workflows werden mit den jeweiligen Parameterwerten gestartet und der Wissenschaftler

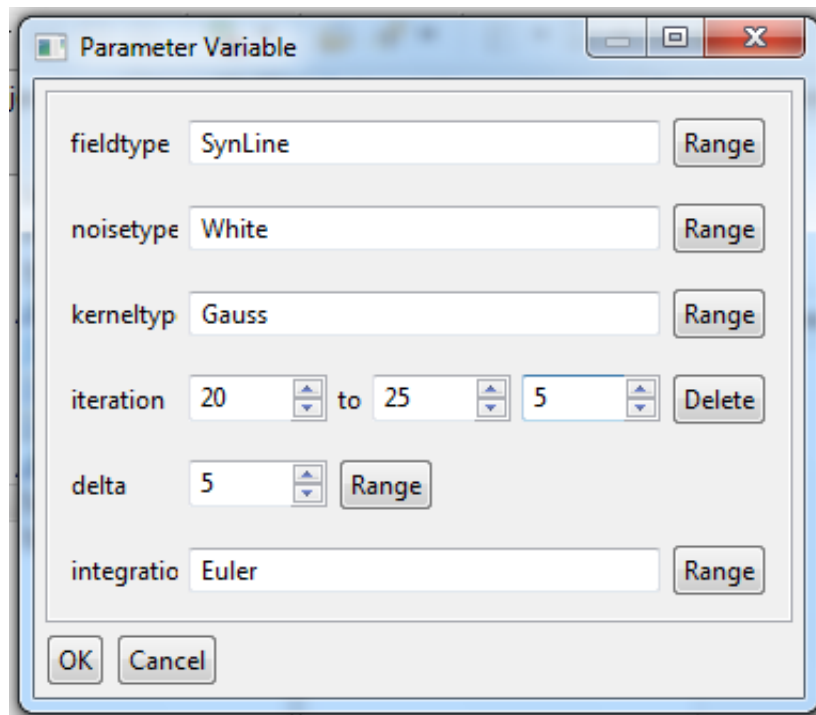


Abbildung 6.9.: Parameterdialog zur Eingabe der Parameterwerte

Listing 6.16 Zweite Nachricht, die an die Apache ODE gesendet wird

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsdl="de.test/wsdl">
  <soapenv:Header/>
  <soapenv:Body>
    <wsdl:testParameterRequest>
      <wsdl:fieldtype>SynLine</wsdl:fieldtype>
      <wsdl:noisetype>White</wsdl:noisetype>
      <wsdl:kerneltype>Gauss</wsdl:kerneltype>
      <wsdl:iteration>25</wsdl:iteration>
      <wsdl:delta>5</wsdl:delta>
      <wsdl:integrationsmethod>Euler</wsdl:integrationsmethod>
    </wsdl:testParameterRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

6. Implementierung

kann nun die gestarteten Workflows beobachten und zwar nicht nur einen Workflow, sondern mehrere parallel, wie die Abbildung 6.10 zeigt.



Abbildung 6.10.: Paralleles Überwachen von Workflows

7. Zusammenfassung und Ausblick

Wissenschaftliche Workflows zeigen viele Gemeinsamkeiten mit herkömmlichen Workflows auf. Beide Workflowtypen haben einen Lebenszyklus, der sich jedoch in einigen Punkten unterscheidet. Wissenschaftler haben eine andere Herangehensweise an Probleme. Sie arbeiten explorativ. Das heißt, sie arbeiten mit Problemen, über die sie noch wenig Wissen besitzen. Um dieses Wissen zu erlangen, führen sie Experimente durch. Solche Experimente können Simulationen oder Berechnungen sein. Auf diesem Gebiet gibt es viel Forschung, um die Workflow Technologie aufgrund ihrer vielen Vorteile auch für wissenschaftliche Simulationen und Experimente einzusetzen.

Diese Arbeit beschäftigte sich mit der konzeptionellen Erweiterung einer bestehenden Plattform für wissenschaftliche Workflows. Zum einen wurde das parametrisierte Starten von wissenschaftlichen Workflows ermöglicht. Dadurch können Wissenschaftler Parameter eingeben und somit ihre Workflows starten. Das gibt ihnen die Möglichkeit Parameterstudien durchzuführen und Experimente durch Workflows simulieren zu lassen, um somit mehr Wissen über ein Problem zu erhalten.

Die zweite Aufgabe bestand darin, den Eclipse BPEL Designer so anzupassen, dass das parallele Beobachten von wissenschaftlichen Workflows möglich wird. Der Eclipse BPEL Designer bot zuvor die Möglichkeit an, nur einen Workflow zu beobachten. Nun kann der Wissenschaftler parallel Workflows ausführen und die Ergebnisse für jeden einzelnen Workflow beobachten. Die entwickelten Konzepte wurden prototypisch für den Eclipse BPEL Designer und der Apache ODE umgesetzt.

7.1. Ausblick

Diese Arbeit zeigt ein Konzept und eine prototypische Realisierung zweier Punkte, um die Arbeitsweise von Wissenschaftlern mit Workflows zu verbessern und zu vereinfachen. Im Zuge dieser Arbeit sind weitere Ideen entstanden, die zur Weiterentwicklung der bestehenden Werkzeuge beitragen können.

Das in dieser Arbeit entwickelte Konzept zum parametrisierten Start von Workflows beschränkt sich momentan nur auf vier Datentypen. Hier wäre eine Analyse sehr hilfreich, welche Datentypen man noch hinzufügen könnte. Eine Möglichkeit zur Erweiterung bietet die Diplomarbeit von [Hot10]. Im Zuge der Diplomarbeit von Hotta wurde ein Ressourcen Management realisiert, das die Verwaltung von Servern, wissenschaftlichen Services und Simulationsdaten übernimmt. Das Ressourcen Management ermöglicht es, Simulations- und

Berechnungsergebnisse logisch und zentralisiert abzuspeichern und zu einem späterem Zeitpunkt wieder zu verwenden und zwischen verschiedenen Werkzeugen auszutauschen.

Durch die Berechnung des kartesischen Produkt, anhand des Wertebereichs der vom Wissenschaftler spezifizierten Parameter, wird für alle Tupel eine Nachricht erstellt, die zur Erstellung von neuen Workflow Instanzen führt. Hier besteht die Gefahr, dass der BPEL Designer mit der Berechnung der Nachrichten sehr lange braucht, wenn die Anzahl der eingegeben Werte sich erhöht. Je nach Anforderungen bietet es sich an, eine genau Analyse durchzuführen, um zu bestimmen, mit welcher Berechnungsgröße der BPEL Designer maximal umgehen kann. Das gleiche gilt ebenfalls für die Apache ODE, die unter Umständen eine größere Menge von Workflow Instanzen gleichzeitig starten und laufen lassen muss.

Bei der parallelen Beobachtung von wissenschaftlichen Workflows bietet es sich an, die Daten zu persistieren. Das Auditing Tool übernimmt schon die Persistierung, allerdings werden hier noch zwei unterschiedliche Datenpools verwendet. Im Eclipse BPEL Designer wird nicht immer auf den Datenpool des Auditing Tools zugegriffen. Die Erweiterung der Auditing Sicht im Eclipse BPEL Designer, die im Zuge dieser Arbeit realisiert wurde, greift nun auf den Datenpool des Auditing Tools zu. Allerdings wird nur beim Öffnen einer bereits persistierten Instanz auf den Datenpool des Auditing Tools zugegriffen, nicht aber beim Starten einer Instanz. Hier wäre die Überlegung sinnvoll, einen gemeinsam genutzten Datenpool zu verwenden. Somit wären Informationen dauerhaft gespeichert und man könnte dem Wissenschaftler, im Punkt Analyse des Lebenszyklus, noch mehr unter die Arme greifen.

A. Anhang

A.1. WSDL

Listing A.1: WSDL Beispiel Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="de.test/wsdl"
  xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop" name="test"
  targetNamespace="de.test/wsdl">

  <!-- ~~~~~
  TYPE DEFINITION - List of services participating in this BPEL process
  The default output of the BPEL designer uses strings as input and
  output to the BPEL Process. But you can define or import any XML
  Schema type and use them as part of the message types.
  ~~~~~ -->

  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
      elementFormDefault="qualified" targetNamespace="de.test/wsdl">
      <element name="testParameterRequest">
        <complexType>
          <sequence>

            <element name="date" type="dateTime"/>
            <element name="datum" type="dateTime"/>
            <element name="number" type="int"/>
            <element name="boolean" type="boolean"/>
            <element name="text" type="string"/>

          </sequence>

        </complexType>
      </element>
    </schema>
  </types>

  <!-- ~~~~~
  MESSAGE TYPE DEFINITION - Definition of the message types used as
  part of the port type definitions
  ~~~~~ -->

  <message name="testParameterMessage">
    <part element="tns:testParameterRequest" name="parameters"/>
  </message>

  <!-- ~~~~~
  PORT TYPE DEFINITION - A port type groups a set of operations into
  a logical service unit.
```

A. Anhang

```
~~~~~ -->
<!-- portType implemented by the test BPEL process -->
<portType name="testPT">
  <operation name="initiate">
    <input message="tns:testParameterMessage"/>
  </operation>
</portType>

<!-- ~~~~~
      BINDING DEFINITION - A binding prescribes the transport protocol,
      message type and encoding to invoke operations of this service.
      ~~~~~ -->
<binding name="testBinding" type="tns:testPT">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate">
    <soap:operation soapAction="de.test/wsdl/initiate"/>
    <input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>

<!-- ~~~~~
      SERVICE DEFINITION - A service contains the location of the service
      (port) and refers to a binding.
      ~~~~~ -->
<service name="testService">
  <port binding="tns:testBinding" name="testPort">
    <soap:address location="http://localhost:8080/ode/processes/testService"/>
  </port>
</service>

<!-- ~~~~~
      PARTNER LINK TYPE DEFINITION
      the test partnerLinkType binds the provider and
      requester portType into an asynchronous conversation.
      ~~~~~ -->
<plnk:partnerLinkType name="testPLT">
  <plnk:role name="testProvider" portType="tns:testPT"/>
</plnk:partnerLinkType>
</definitions>
```

A.2. SOAP Nachrichten

Für die Berechnung der Nachrichten wird das kartesische Produkt herangezogen. In diesem Beispiel existieren die fünf Parametervariablen *date*, *datum*, *number*, *boolean* und *text*. *Date* und *Datum* sind vom Typ *DateTime*, *number* vom Typ *Integer*, *boolean* vom Typ *Boolean* und *text* vom Typ *String*. Dabei besitzen die Parametervariablen folgende Werte:

- *Date* = {2011 – 10 – 19}
- *Datum* = {2011 – 10 – 31}

Listing A.1 Erste SOAP Nachricht aus dem kartesischem Produkt

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wSDL="de.test/wSDL">
  <soapenv:Header/>
  <soapenv:Body>
    <wSDL:testParameterRequest>
      <wSDL:date>2011-10-19</wSDL:date>
      <wSDL:datum>2011-10-31</wSDL:datum>
      <wSDL:number>498</wSDL:number>
      <wSDL:boolean>true</wSDL:boolean>
      <wSDL:text>Das ist ein Beispiel für eine Nachricht</wSDL:text>
    </wSDL:testParameterRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing A.2 Zweite SOAP Nachricht aus dem kartesischem Produkt

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wSDL="de.test/wSDL">
  <soapenv:Header/>
  <soapenv:Body>
    <wSDL:testParameterRequest>
      <wSDL:date>2011-10-19</wSDL:date>
      <wSDL:datum>2011-10-31</wSDL:datum>
      <wSDL:number>30</wSDL:number>
      <wSDL:boolean>true</wSDL:boolean>
      <wSDL:text>Das ist ein Beispiel für eine Nachricht</wSDL:text>
    </wSDL:testParameterRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

- $Number = \{498, 30\}$
- $Boolean = \{true\}$
- $Text = \{Das\ ist\ ein\ Beispiel\ für\ eine\ Nachricht\}$

Nach der Berechnung des kartesischen Produkts werden folgende zwei SOAP Nachrichten erstellt:

A.3. BPEL

Listing A.2: BPEL Datei

```
<!-- SampleProcess BPEL Process [Generated by the Eclipse BPEL Designer] -->
<!-- Date: Mon Oct 24 14:33:22 CEST 2011 -->

<bpel:process name="ShowLic"
    targetNamespace="de.sample.thesis"
    suppressJoinFailure="yes"
    xmlns:tns="de.sample.thesis"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:procwsdl="de.sample.thesis/wsdl"
    xmlns:ns1="http://www.w3.org/2001/XMLSchema" xmlns:ns="http://iaas.ustutt.de">
    <!-- Import the client WSDL -->
    <bpel:import namespace="http://iaas.ustutt.de" location="echo.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
    <bpel:import location="SampleProcessArtifacts.wsdl" namespace="de.sample.thesis/wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/" />

    <!-- ===== -->
    <!-- PARTNERLINKS -->
    <!-- List of services participating in this BPEL process -->
    <!-- ===== -->
    <bpel:partnerLinks>
        <!--
        The 'client' role represents the requester of this service. It is
        used for callback. The location and correlation information associated
        with the client role are automatically set using WS-Addressing.
        -->
        <bpel:partnerLink name="client"
            partnerLinkType="procwsdl:SampleProcessPLT"
            myRole="SampleProcessProvider"
            />
        <bpel:partnerLink name="EchoPL" partnerLinkType="procwsdl:EchoPLT"
            partnerRole="EchoProvider"></bpel:partnerLink>
    </bpel:partnerLinks>

    <!-- ===== -->
    <!-- VARIABLES -->
    <!-- List of messages and XML documents used within this BPEL process -->
    <!-- ===== -->
    <bpel:variables>
        <!-- Reference to the message passed as input during initiation -->
        <bpel:variable name="input"
            messageType="procwsdl:SampleProcessParameterMessage"/>
        <bpel:variable name="fielddtype" type="ns1:string" parameter="yes"></bpel:variable>
        <bpel:variable name="noisettype" type="ns1:string" parameter="yes"></bpel:variable>
        <bpel:variable name="kerneltype" type="ns1:string" parameter="yes"></bpel:variable>
        <bpel:variable name="iteration" type="ns1:int" parameter="yes"></bpel:variable>
        <bpel:variable name="delta" type="ns1:int" parameter="yes"></bpel:variable>
        <bpel:variable name="integrationsmethod" type="ns1:string"
            parameter="yes"></bpel:variable>
        <bpel:variable name="EchoPLResponse" messageType="ns:echoResponse"></bpel:variable>
        <bpel:variable name="EchoPLRequest" messageType="ns:echoRequest"></bpel:variable>
    </bpel:variables>
</bpel:process>
```

```

</bpel:variables>

<!-- ===== -->
<!-- ORCHESTRATION LOGIC -->
<!-- Set of activities coordinating the flow of messages across the -->
<!-- services integrated within this business process -->
<!-- ===== -->
<bpel:sequence name="main">

    <!-- Receive input from requestor.
        Note: This maps to operation defined in SampleProcess.wsdl
        -->
    <bpel:receive name="receiveInput ( fieldtype, noisetype, kerneltype, iteration,
        delta, integrationsmethod)" partnerLink="client"
        portType="procwsdl:SampleProcessPT"
        operation="initiate" variable="input"
        createInstance="yes"/>

    <bpel:assign validate="yes" name="Assign (fieldtype,noisetype,kerneltype)">
</bpel:assign>
    <bpel:flow name="Flow"><bpel:invoke name="loadVector" partnerLink="EchoPL"
        operation="echo"
            portType="ns:EchoServicePortType" inputVariable="EchoPLRequest"
            outputVariable="EchoPLResponse">
        </bpel:invoke></bpel:flow>

    <bpel:assign validate="no" name="Assign (vecor, noise, kernel, iteration, delta,
        integrationsmethod)">
    <bpel:copy>
        <bpel:from>
            <bpel:literal xml:space="preserve">
                <ns:echo xmlns:ns="http://iaas.ustutt.de"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    </ns:echo>
                </bpel:literal>

            </bpel:from>
            <bpel:to variable="EchoPLRequest" part="parameters"></bpel:to>
        </bpel:copy>
    <bpel:copy>
        <bpel:from part="parameters" variable="input">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                <![CDATA[procwsdl:noisetype]]>
            </bpel:query>

        </bpel:from>
        <bpel:to part="parameters" variable="EchoPLRequest">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                <![CDATA[ns:hello]]>
            </bpel:query>

        </bpel:to>
    </bpel:copy>
</bpel:assign>
    <bpel:invoke name="LIC calculation" partnerLink="EchoPL" operation="echo"
        portType="ns:EchoServicePortType" inputVariable="EchoPLRequest"
        outputVariable="EchoPLResponse">
    </bpel:invoke>

```

```

    <bpel:assign validate="no" name="Assign calculation return">
      <bpel:copy>
        <bpel:from>
          <bpel:literal xml:space="preserve">
            <ns:echo xmlns:ns="http://iaas.ustutt.de"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
              </ns:echo>
            </bpel:literal>
          </bpel:from>
          <bpel:to variable="EchoPLRequest" part="parameters"></bpel:to>
        </bpel:copy>
        <bpel:copy>
          <bpel:from part="parameters" variable="input">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
              <![CDATA[procwsdl:integrationsmethod]]>
            </bpel:query>
          </bpel:from>
          <bpel:to part="parameters" variable="EchoPLRequest">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
              <![CDATA[ns:hello]]>
            </bpel:query>
          </bpel:to>
        </bpel:copy>
      </bpel:assign>
      <bpel:invoke name="show image in GUI" partnerLink="EchoPL" operation="echo"
        portType="ns:EchoServicePortType" inputVariable="EchoPLRequest"
        outputVariable="EchoPLResponse">
      </bpel:invoke>
      <bpel:assign validate="no" name="Assign return value"></bpel:assign>
    <!--      <bpel:reply name="replyOutput" partnerLink="client" operation="initiate"
      portType="procwsdl:SampleProcessPT"></bpel:reply> -->
  </bpel:sequence>
</bpel:process>

```

Literaturverzeichnis

- [ABJ⁺04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock. Kepler: An Extensible System for Design and Execution of scientific workflows. *Scientific and Statistical Database Management*, pp. 423–424, 2004. (Zitiert auf Seite 35)
- [BG07] R. Barga, D. B. Gannon. Scientific versus Business Workflows. In *Workflows for e-Science*, pp. 9–16. Springer, 2007. (Zitiert auf Seite 16)
- [BJA⁺07] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, Y. Simmhan. The Trident Scientific Workflow Workbench. *IEEE International Conference on eScience*, 2007. (Zitiert auf Seite 35)
- [Bono8] F. Bongers. *XSLT 2.0 & XPath 2.0*. Galileo Computing, 2008. (Zitiert auf Seite 24)
- [CFNO02] M. Champion, C. Ferris, E. Newcomer, D. Orchard. Web Services Architecture. Technical report, W3C, 2002. (Zitiert auf den Seiten 20, 21 und 22)
- [CGH⁺06] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, I. Wang. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation: Practice and Experience. Special Issue: Workflow in Grid Systems*, 18(10):1012–1037, 2006. (Zitiert auf Seite 35)
- [Che09] X. Chen. *Integration of the Result Visualization into a Workflow Modeling Tool*. Master’s thesis, Universität Stuttgart, 2009. (Zitiert auf den Seiten 34, 70 und 71)
- [DBG⁺04] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny. Pegasus: Mapping Scientific Workflows onto the Grid. *Lecture Notes in Computer Science*, 3165:131–140, 2004. (Zitiert auf Seite 35)
- [Eic10] A. Eichel. *Interaktives Monitoring von wissenschaftlichen Workflows*. Master’s thesis, Universität Stuttgart, 2010. (Zitiert auf den Seiten 26, 32, 33, 40, 47, 48 und 50)
- [GHJ09] E. Gamma, R. Helm, R. Johnson. *Entwurfsmuster . Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2009. (Zitiert auf Seite 56)
- [Grao2] S. Graham. *Building Web services with Java : making sense of XML, SOAP, WSDL, and UDDI*. Indianapolis Sams, 2002. (Zitiert auf Seite 21)
- [GSK⁺11] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, M. Reiter. Conventional Workflow Technology for Scientific Simulation. In *Guide to e-Science*, pp. 323–352. Springer, 2011. (Zitiert auf den Seiten 7, 8, 16, 19, 22 und 35)
- [Hot10] S. Hotta. *Ausführung von Festkörpersimulationen auf Basis der Workflow Technologie*. Master’s thesis, Universität Stuttgart, 2010. (Zitiert auf Seite 77)

- [KKS⁺06] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszczek, F. Leymann. BPEL Event Model. Technical report, Institut für Architektur von Anwendungssystemen, 2006. (Zitiert auf den Seiten 30, 31, 32 und 33)
- [LLN11] T. van Lessen, D. Lübke, J. Nitzsche. *Geschäftsprozesse automatisieren mit BPEL*. dpunkt, 2011. (Zitiert auf den Seiten 23 und 24)
- [LRoo] F. Leymann, D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall, 2000. (Zitiert auf den Seiten 7, 9, 11, 12, 13 und 23)
- [LWMB09] B. Ludäscher, M. Weske, T. McPhillips, S. Bowers. Scientific Workflows: Business as Usual? In *Proceedings of the International Conference on Business Process Management*, pp. 31–47. Springer, 2009. (Zitiert auf den Seiten 7, 16 und 17)
- [MWo6] G. Merziger, T. Wirth. *Repetitionum der höheren Mathematik*. Binomi, 2006. (Zitiert auf Seite 44)
- [Nito6] J. Nitzsche. *Entwicklung eines Monitoring-Tools zur Unterstützung von parametrisierten Web Service Flows*. Master's thesis, Universität Stuttgart, 2006. (Zitiert auf den Seiten 33 und 34)
- [ODE] A. ODE. ODE - Architectural Overview. URL <http://ode.apache.org/architectural-overview.html>. (Zitiert auf den Seiten 28 und 29)
- [OGA⁺06] T. Oinn, M. Greenwood, M. Addis, M. N. Apldemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. S. and Robert Stevens, A. Wipat, C. Wroe. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience. Special Issue: Workflow in Grid Systems*, 18(10):1067–1100, 2006. (Zitiert auf Seite 35)
- [Org07] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language Version 2.0*, 2007. (Zitiert auf den Seiten 25 und 26)
- [SBPM09] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2009. (Zitiert auf Seite 27)
- [SK10] M. Sonntag, D. Karastoyanova. Next Generation Interactive Scientific Experimenting Based on the Workflow Technologies. In *Proceedings of the 21st IASTED International Conference Modelling and Simulation*. MS 2010, 2010. (Zitiert auf den Seiten 14, 15, 17, 18 und 19)
- [SKL10] M. Sonntag, D. Karastoyanova, F. Leymann. The Missing Features of Workflow Systems for Scientific Computations. In *3rd Grid Workflow Workshop*. Gesellschaft für Informatik e.V., 2010. (Zitiert auf den Seiten 14, 16 und 18)
- [SSoo] K. Scribner, M. C. Stiver. *Understanding SOAP*. Indianapolis Sams, 2000. (Zitiert auf Seite 21)
- [STo7] G. Starke, S. Tilkov. *SOA - Expertenwissen*. dpunkt, 2007. (Zitiert auf Seite 19)

- [Steo8] T. Steinmetz. *Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE*. Master's thesis, Universität Stuttgart, 2008. (Zitiert auf den Seiten 30 und 33)
- [TDGSo7] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields. *Workflow for e-Science*. Springer, 2007. (Zitiert auf den Seiten 15 und 16)
- [TSWHo7] I. Taylor, M. Shields, I. Wang, A. Harrison. The Triana Workflow Environment: Architecture and Applications. In *Workflow for e-Science*. 2007. (Zitiert auf Seite 35)
- [Utko8] E. Utkin. *A Monitoring Tool for the Apache ODE BPEL Engine*. Master's thesis, Universität Stuttgart, 2008. (Zitiert auf Seite 34)
- [WCL⁺05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. (Zitiert auf den Seiten 20, 21, 22, 24 und 25)
- [WEB⁺07] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, J. Patel. Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling. In *Workflows for e-Science*. 2007. (Zitiert auf Seite 34)
- [Wes07] M. Weske. *Business Process Management*. Springer, 2007. (Zitiert auf den Seiten 9, 11 und 15)

Alle URLs wurden zuletzt am 31.10.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Aleksandar Tolev)