Fakultät Informatik, Elektrotechnik und Informationstechnik
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2328

Kombination von SIMPL mit einem Ansatz zur Propagation von Datenänderungen

Christian Ageu

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Peter Reimann
begonnen am:	31. März 2011
beendet am:	30. September 2011
CR-Klassifikation:	D.2.11, H.2.5, H.3.4, H.4.1, I.6.7

Inhaltsverzeichnis

1 Einführung **7**

2 Grundlagen **9**
2.1 Extensible Markup Language 9
2.2 SOA und Webservices 9
2.3 Workflows und Workflowmanagementsysteme 10
2.4 Web Services Business Process Execution Language 12
2.5 Simulationsworkflows 13
2.5.1 Das PANDAS-Rahmenwerk 13
2.5.2 DM-Problematik 14

3 SimTech - Information Management, Processes and Languages **17**
3.1 Architektur 18
3.2 DM-Aktivitäten 19
3.3 DM-Patterns 20

4 Champagne **23**
4.1 Anomalien bei Datenänderungen 23
4.2 Basiskonzepte 24
4.2.1 Abhängigketen und Änderungsbeschreibungen 24
4.2.2 Propagationsskripte 25
4.3 Architektur 26
4.3.1 Repository 26
4.3.2 Abhängigkeitsmanager (Dependency Manager) 28
4.3.3 Propagationsmanager (Propagation Manager) 28
4.3.4 Adapter (Adaptors) 29

5 Kombination der Technologien **31**
5.1 Gemeinsamkeiten und Unterschiede 31
5.1.1 Data Source Connector, Data Converter und Adapter 31
5.1.2 Resource Management und Repository 32
5.1.3 DM Activity Modeling Plug-In und Abhängigkeitsmanager 32
5.1.4 Laufzeitkomponenten 33
5.2 Champagne als Erweiterung von SIMPL 33
5.3 Champagne als Dienst 34
5.4 Champagne als Ergänzung eines Workflows 34
5.5 SIMPL zur Datenänderungspropagation 35

6 Implementierung **37**
6.1 Anforderungen 37
6.2 Metadaten 38
6.3 Entwurfs- und Laufzeitkomponenten 38
6.4 Funktionsweise des Transferpattern-Managers 39

7 Ergebnis und zukünftige Arbeiten **41**

Literaturverzeichnis **43**

Abbildungsverzeichnis

2.1 Architektur eines sWfMS nach [G+11] 11
2.2 PANDAS-Workflow nach [RRS+11] 14
2.3 Beispiel für Datenquellen/-formate im PANDAS-Wf 15

3.1 SIMPL-Rahmenwerk integriert in ein sWfMS (Vgl. [RRS+11]) 18
3.2 Links: Verwendung einer SIMPL DM-Aktivität innerhalb eines Workflows
Rechts: Aufruf eines gekapselten Dienstes, der eine DM-Aktivität beinhaltet. . 19
3.3 Join-Pattern und seine ausführbaren Umformungen gemäß [RRS+11] 21

4.1 Anomalie bei Änderungen mehrfach repräsentierter Information 24
4.2 Architektur von Champagne. [RCHM02] 27

6.1 Architektur des Transferpattern-Managers (Vgl. Abbildung 3.1) 39

1 Einführung

Zur Beschreibung und Modellierung von Prozessen haben sich in den vergangenen Jahren Workflows etabliert [LR00]. Sowohl im Produktionsumfeld [MRZ11] als auch bei computergestützten Simulationen [TDG07] werden Vorgänge zunehmend durch Workflows beschrieben. Dabei werden verschiedene Teilprozesse in ein Ablaufschema überführt, welches den Gesamtprozess als Ausführungsgraph dieser Teilprozesse modelliert. Der Gesamtprozess kann dann als gesonderter Arbeitschritt aufgefasst und wiederum selbst in größere Abläufe integriert werden.

Oft ist jeder Teilprozess ein in sich geschlossener, eigenständiger Vorgang. Die Verknüpfung aufeinanderfolgender Vorgänge ist nur möglich, wenn alle Voraussetzungen für den nachfolgenden Vorgang von den vorangegangenen erfüllt werden [LR00]. So kann beispielsweise in der industriellen Fertigung die Weiterverarbeitung eines Werkstücks nur dann erfolgen, wenn durch die vorherigen Arbeitsschritte gewährleistet wurde, dass es in der richtigen Form vorliegt. Bei datenverarbeitenden Prozessen trifft dieser Sachverhalt für die jedem Teilprozess zugrundeliegende Datenbasis zu. Die von Prozessen erzeugten Daten müssen von den weiterverarbeitenden Nachfolgeprozessen gelesen und richtig interpretiert werden können [RRS+11].

Diese Arbeit beschäftigt sich vorwiegend mit Simulationsprozessen und damit verbundenen Vorgängen und Werkzeugen. Es wird anhand von Beispielen auf momentan vorherrschende Problemstellungen beim Datenmanagement (DM) solcher Workflows mit der *Business Process Execution Language* (BPEL) [bpe] hingewiesen. Die bereits existierenden DM-Lösungskonzepte aus unterschiedlichen Bereichen *SIMPL* [RRS+11] und *Champagne* [Hei11] werden vorgestellt und untersucht, in wie weit sich Bestandteile des einen Systems in das andere übernehmen lassen. Hauptziel ist dabei die Untersuchung der Automatisierbarkeit von Datentransfers zwischen verschiedenen Resourcen aus einem Workflow heraus, um die Datenkompatibilität zwischen Simulationsprozessen mit unterschiedlichen (heterogenen) Daten zu gewährleisten.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden grundlegende Begriffe erläutert und die in der Arbeit verwendeten Technologien vorgestellt. Es wird außerdem ein Anwendungsfall und dessen Ausgangsproblematik beschrieben.

Kapitel 3 – SimTech - Information Management, Processes and Languages: Das Datenmanagement-Rahmenwerk für Workflows wird vorgestellt und analysiert.

Kapitel 4 – Champagne: Der Propagationsmanager für Datenänderungen in Informationssystemen aus dem Produktionsumfeld wird vorgestellt und analysiert.

Kapitel 5 – Kombination der Technologien: Die beiden Systeme *SIMPL* und *Champagne* werden gegenübergestellt und ihre Komponenten bzw. Konzepte auf Kombinationsmöglichkeiten hin untersucht.

Kapitel 6 – Implementierung: Anhand der Erkenntnisse in Kapitel 5 wird eine mögliche Implementierung einer Komponente untersucht, die die Lösung einer konkreten Problemstellung des Anwendungsfalles umsetzt.

Kapitel 7 – Ergebnis und zukünftige Arbeiten: Es werden abschließende Kommentare abgegeben und auf weitere Themen hingewiesen, die sich im Verlauf der Arbeit aufgetan haben, jedoch nicht behandelt wurden.

2 Grundlagen

In diesem Kapitel werden die Grundbegriffe und Werkzeuge vorgestellt, die entweder bei der Modellierung von Workflows im allgemeinen eine wichtige Rolle spielen oder im weiteren Verlauf dieser Arbeit benötigt werden. Dabei werden die ersten Probleme und Konflikte vorgestellt, für die im Rahmen dieser Arbeit Lösungen gefunden werden sollen.

2.1 Extensible Markup Language

Die e**X**tensible **M**arkup **L**anguage [xml], kurz XML, ist eine weit verbreitete Sprache zur Darstellung von hierarchisch strukturierten Daten in Textform. Ein XML-Dokument besteht zunächst aus genau einem Elelement, dem Wurzelelement. Dieses kann Text und weitere Unterelemente beinhalten. Bei der repräsentation von Daten mit XML wird Text als Datum üblicherweise nur in Blattlelementen verwendet, wobei der Elementpfad vom Wurzelknoten aus die Bezeichnung liefert. Die Struktur eines XML-Dokuments und der Inhalt der Texteinträge können dabei in einer separaten *XML Schema Definition*(XSD)-Datei definiert werden.

XML bietet umfangreiche Möglichkeiten zur Verwaltung, Abfrage und Manipulation von Daten. Mit XQuery [xqu] und XSLT [xsl] stehen zwei mächtige Sprachen zur Abfrage bzw. Transformation von XML-Daten zur Verfügung. Diese beide Sprachen verwenden dabei XPath [xpa], um auf einzelne Elemente eines XML-Dokuments zuzugreifen.

Durch den hohen Verbreitungsgrad von XML existieren für diese Sprachen viele Werkzeuge, mit der sich eine XML-basierte Datenbasis realisieren lässt. Sowohl die Beschreibungssprache für Workflows (BPEL), als auch die in dieser Arbeit untersuchten Werkzeuge SIMPL und Champagne benutzen XML als Grundlage zum Austausch und zur Speicherung ihrer Daten.

2.2 SOA und Webservices

Serviceorientierte Architektur (SOA) ist ein Architekturmuster, bei dem Geschäftsprozesse mit Hilfe von Diensten realisiert werden. Anstatt einen Gesamtprozess als Ganzes zu implementieren, wird versucht, ihn in kleinere Teilprozesse aufzuspalten. Ein so gekapselter Teilprozess bildet einen *Dienst*, der isoliert ausgeführt eine bestimmte Aufgabe erfüllt. Mehrere Dienste zusammen bilden bei koordinierter (*orchestrierter*) Ausführung den Gesamtprozess. Dem zugrunde liegt ein System aus *Konsument*, *Anbieter* und einem *Verzeichnis*. Besonderes

Merkmal dieser Architektur ist die lose Kopplung der Dienste an die aufrufenden Prozesse: Ein Konsument ist nicht an einen bestimmten Dienst gebunden, sondern findet über das Verzeichnis anhand einer Beschreibung einen Dienst, der die benötigte Funktionalität zur Verfügung stellt. Der besondere Vorteil bei dieser Architektur liegt in der Wiederverwendbarkeit der einzelnen Dienste. Die Unterteilung von Geschäftsprozessen in gekapselte Dienste bietet auch eine höhere Überschaubarkeit und damit einfachere Entwicklung und Pflege der Prozesse.

Das Konzept der *Webservices* ermöglicht es, fertiggestellte Dienste plattformunabhängig in einem Netzwerk über das HTTP-Protkoll bereitzustellen. Möchte ein Anbieter einen Dienst als Webservice zur Verfügung stellen, so meldet er den Dienst und dessen Schnittstellen, beschrieben in der XML-basierten *Web Services Description Language* (WSDL), bei einem Verzeichnisserver, der sog. UDDI Registry (*Universal Description, Discovery, and Integration*), an. Ein interessierter Konsument kann diesen daraufhin über die UDDI Registry finden (*discover*), und ihn anschließend aufrufen (*invoke*). Der Datenaustausch zwischen Anbieter und Konsument erfolgt mittels ebenfalls XML-basierter SOAP-Nachrichten (*Simple Object Access Protocol* [soa]). Die Rollen des Anbieters und des Konsumenten übernehmen dabei die Prozesse, die den Dienst bereitstellen bzw. benötigen. Da die aufrufenden Prozesse auch selbst Webservices sein können, wird mit Hilfe dieses Konzeptes eine hohe Kombinationsmöglichkeit und Wiederverwendung von Diensten ermöglicht.

2.3 Workflows und Workflowmanagementsysteme

Ein *Workflow* (Arbeitsablauf) ist die Verknüpfung verschiedener Arbeitsschritte zu einem Gesamtablauf, basierend auf kausalen Abhängigkeiten oder Datenabhängigkeiten zwischen den Teilschritten. Die Repräsentation von Workflows erfolgt entweder mit Hilfe von Diagrammen oder durch eine *Workflowsprache*, die einen Prozess in maschinenlesbarer Form beschreibt. Ein Beispiel für eine solche Sprache ist die *XML Process Definition Language* (XPDL) [xpd]. Wir betrachten Workflows, die vollständig durch ein *Workflowmanagementsystem* (WfMS) auf einem Computer ausgeführt werden.

In der Vergangenheit wurden Workflows hauptsächlich zur EDV-gestützten Beschreibung und Ausführung wirtschaftlicher Geschäftsprozesse verwendet (*business workflows*). Seit kurzem finden Workflows auch im wissenschaftlichen Bereich Anwendung (*scientific workflows [TDG07]*), insbesondere im Gebiet der Simulationen [G+11]. Die dabei eingesetzten Workflowsysteme werden *scientific Workflow Management Systems*, kurzsWfMS, genannt.

Abbildung 2.1 veranschaulicht die Architektur eines sWfMS basierend auf der in [LR00] definierten Technologie für Geschäfts- und Produktionsworkflows. Es folgt eine Beschreibung der einzelnen Komponenten gemäß [RRS+11] und [G+11].

Zuerst werden die Komponenten der **GUI** betrachtet:

- Der **sWf Modeler** unterstützt den Entwickler beim Modellieren der Spezifikationen und Deploymentinformationen eines Workflows.

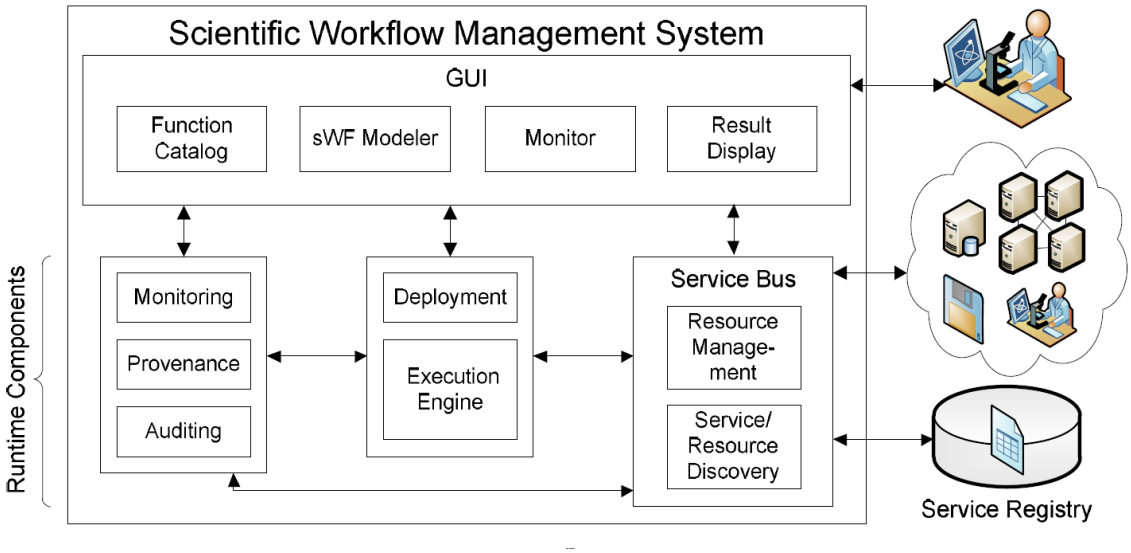

Abbildung 2.1: Architektur eines sWfMS nach [G+11]

- Der **Function Catalog** stellt eine Liste von im Workflow verfügbaren Diensten bereit, die bei der Modellierung eingesetzt werden können.
- Die **Monitor**-Komponente bietet eine Benutzerschnittstelle, die zur Überwachung der Ausführung von Workflow-Instanzen und damit zum Erkennen von unerwarteten Ereignissen bzw. Fehlern während der Ausführung dient.
- Das **Result-Display** liefert die Zwischen- und Endergebnisse des ausgeführten Workflows (z.B. Simulationsergebnisse) in einem für den Benutzer bedarfsgerechten Format.

Als nächstes werden die Laufzeitkomponenten (Runtime Components) erläutert:

- Die **Deployment**-Komponente überführt das Modell eines Workflows in ein Objekt und installiert dieses in einer **Ausführungsengine** (z.B Apache ODE[1]), die später Instanzen davon ausführt.
- Die **Auditing**-Komponente speichert workflowbezogene Ereignisse und Aktivitäten, die zur Laufzeit auftreten, z.B. den Anfangszeitpunkt eines Workflowaufrufs.
- Die **Monitoring**-Komponente überwacht die Zustände von Workflowausführungen mit Hilfe der Daten aus der Auditing-Komponente.
- Die **Provenance**-Komponente erfasst weitere, detaillierte Daten einer Ausführung als die Auditing-Komponente. Mit Hilfe dieser Informationen lassen sich Ausführungen exakt nachvollziehen.

[1]http://ode.apache.org/

- Der **Service Bus** ist für das Finden und Auswählen von Diensten zur Implementierung des Workflows zuständig. Desweiteren dient er der Zustellung von Nachrichten sowie der Durchführung von Datentransformationen innerhalb des sWfMs und kann externe Ressourcen (z.B. Datenquellen) an den Workflow anbinden.
- Die **Resource Management**-Komponente speichert Meta-Informationen über die externen Ressourcen und Dienste.
- Die **Service/Resource Discovery**-Komponente erstellt anhand dieser Meta-Informationen oder mit Hilfe externer Verzeichnisse eine Liste der in Frage kommenden Dienste und Ressourcen, die vorher beispielsweise durch *semantische Annotation* beschrieben wurden (vgl. lose Kopplung, Abschnitt 2.2). Mit Hilfe dieser Liste ist während der Ausführung beipielsweise im Fehlerfall die Anbindung eines alternativen Dienstes oder einer alternativen Ressource möglich.

2.4 Web Services Business Process Execution Language

Die **W**eb **S**ervices **B**usiness **P**rocess **E**xecution **L**anguage [bpe], kurz: BPEL, ist eine XML-basierte Workflowsprache, die Vorgänge innerhalb von Geschäftsprozessen mit Hilfe von Webservices beschreibt. Sie gilt als de-facto Standard zur Implementierung von Geschäftsprozessen. Mit ihrer Hilfe können Workflows beschrieben, bearbeitet und ausgetauscht werden. Sie ermöglicht eine integrierte und selbstständige Ausführung des Workflows durch ein WfMS, sofern alle Teilprozesse und die Bereitstellung der dazu benötigten Daten ebenfalls automatisch durchgeführt werden können. Bei der Beschreibung von Workflows mit Hilfe von BPEL werden im Allgemeinen alle Teilschritte durch Webservices implementiert. Eine Ausnahme hiervon bildet z.B. die Erweiterung *BPEL4People* [KKL+05], bei der menschliche Interaktion in einen Workflow eingebettet werden kann. Die wesentlichen Vorteile von BPEL bei der Beschreibung von wissenschaftlichen Workflows sind der modulare Aufbau, die Flexibilität im Umgang mit genereischen XML-Datentypen und der späten Anbindung von Diensten an den Workflow, sowie umfangreiche Möglichkeiten zu Fehlerbehandlung und -kompensation [AMA06].

Grundlage der Ausführung von Workflows mit BPEL sind *Aktivitäten*. Eine Aktivität repräsentiert einen Ausführungsschritt innerhalb eines Workflows. Es gibt drei grundlegende Arten von BPEL-Aktivitäten:

- **Aktionen** führen einen bestimmten Arbeitsschritt aus. Dazu gehören der Empfang einer Nachricht mit *Receive*, die Rückmeldung an einen übergeordneten Prozess mit *Reply*, das Aufrufen eines Webservice mit *Invoke* und die Manipulation workflowinterner Variablen durch (*Assign*).
- **Kontrollstrukturen** dienen der bedingten oder wiederholten Ausführung bestimmter Teile eines Workflows sowie der Organisation von Aktivitäten. Mit *If* wird eine odere mehrere Bedingungen geprüft und abhängig vom Ergebnis unterschiedliche Folgeaktivitäten ausgeführt. Mit *While* können Aktivitäten wiederholt ausgeführt werden,

solange die damit verknüpften Bedingungen zutreffen. Mit Hilfe der *Scope*-Aktivität können mehrere Aktivitäten zu einem Verbund zusammengefasst werden. Ein Scope stellt einen Gultigkeitsbereich für Variablen dar und erlaubt die Definition von Fehlerbehandlung und -kompensation für die enthaltenen Aktivitäten. Alle Aktivitäten innerhalb eines Scope können dabei wieder rückgängig gemacht werden, ohne dass Aktivitäen außerhalb des Scope davon betroffen sind.

- **Fehlerbehandlung** kann durch das Werfen von Ausnahmen (*Throw*) kontrolliert werden. Daraufhin kann etwa die Ausführung eines Prozesses abgebrochen werden (*Exit*). write more

Über eine *Extension Activity* kann die Sprache um neue, selbstdefinierte Aktivitäten erweitert werden, z.B um BPEL4People zu realisieren. Das SIMPL-Rahmenwerk fügt der BPEL-Palette auf diese Weise einige neue Aktivitäten für den Zugriff auf und die Verarbeitung von externen Daten hinzu. Diese werden in Kapitel 3 dieser Arbeit näher erläutert.

2.5 Simulationsworkflows

Ein Simulationsworkflow beschreibt Simulationsabläufe, die hauptsächlich auf EDV-Infrastrukturen stattfinden. Bei der Durchführung werden die beteiligten Simulationsprogramme in einen Workflow beschrieben und von einem sWfMS aufgerufen. Die meist schon lange vorhandenen Programme müssen dabei vorher in Form von Webservices bereitgestellt werden, damit sie vom sWfMS eingebunden werden können. Bei einem Simulationsworkflow werden üblicherweise sehr große Datenmengen von verschiedenen Programmen aus teilweise unterschiedlichen Fachgebieten bereitgestellt, bevor sie schließlich durch den eigentlichen Simulationsvorgang verarbeitet werden.

2.5.1 Das PANDAS-Rahmenwerk

Um die Zusammenhänge zu veranschaulichen, wird in Anlehnung an [RRS+11] das PANDAS-Rahmenwerk betrachtet (Porous Media Adaptive Nonlinear finite element solver based on Differential Algebraic Systems)[2]. Es implementiert die Simulation von Knochenverformungen, wie sie bei der orthopädischen Krankheitsforschung eingesetzt wird. Abbildung 2.2 beschreibt den Ablauf eines Simulationsvorganges.

Der Ablauf ist in jeweils einer Preprocessing-, Lösungs- und Postprocessingphase unterteilt. In der *Preprocessing-Phase* werden alle für die Simulation erforderlichen Daten bereitgestellt. Im ersten Schritt werden die Grundinformationen, wie die Struktur des betrachteten Knochens, aus externen Datenbanken oder Dateisystemen geladen. Danach werden Parameter für die Finite-Elemente-Methode, etwa die Wahl der Interpolationsfunktionen, aus vorher angelegten Dateien ausgelesen. Im nächsten Schritt werden Anfangs- und Randbedingungen

[2]http://www.get-pandas.com/

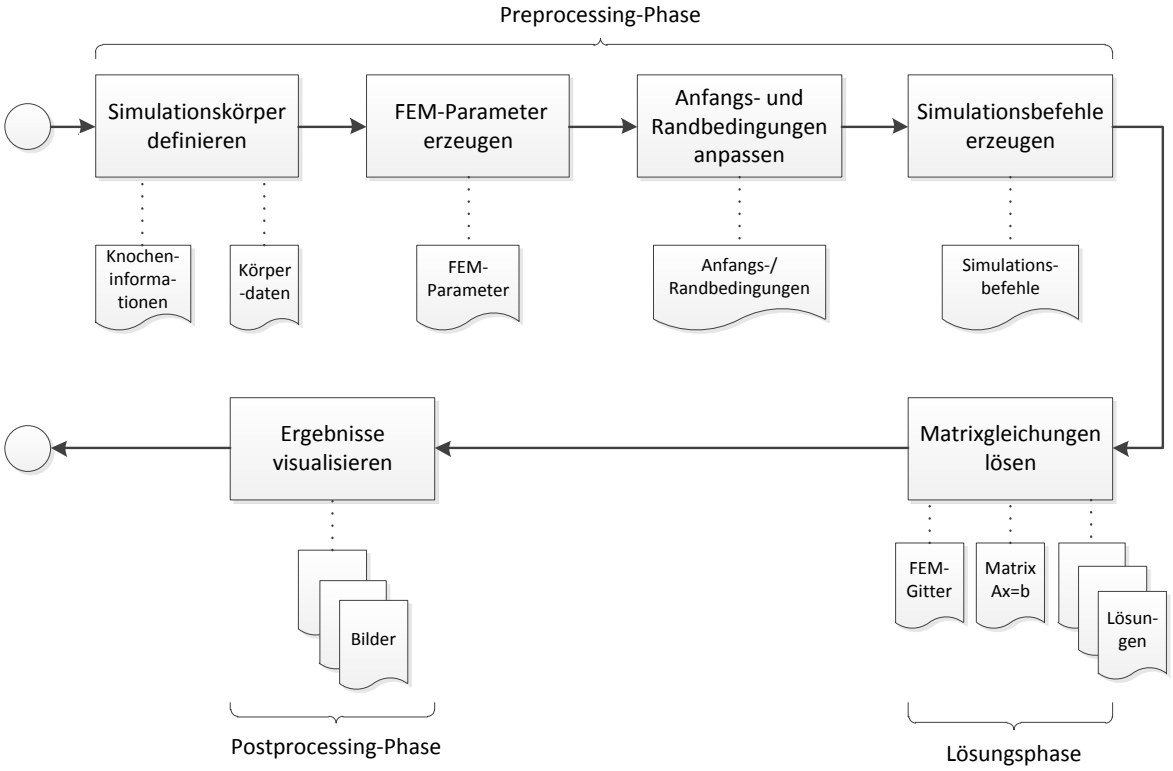

Abbildung 2.2: PANDAS-Workflow nach [RRS+11]

des Verformungsvorganges für den späteren Simulationsprozess angepasst. Im letzten Schritt der Preprocessing-Phase werden schließlich noch Simulationsbefehle erzeugt, die z.B. die Simulationsdauer und die Einteilung dieser in diskrete Zeitschritte bestimmen.

In der **Lösungsphase** werden alle in der Preprocessing-Phase gesammelten Informationen verwendet und mit Hilfe der Finite Elemente Methode (FEM) die jeweiligen Lösungen der Matrizengleichungen für alle festgelegten Zeitschritte ermittelt. Dabei entstehen Gitternetze mit sehr vielen Knotenpunkten, die ebenfalls gespeichert und verwaltet werden müssen. Nachdem die Lösungen für alle eingestellten Zeitschritte berechnet wurden, werden diese Informationen im *CSV*-Format (*C*omma *S*eperated *V*alues) gespeichert und schließlich in der *Postprocessing-Phase* von einem Visualisierungstool in grafische Ausgaben umgesetzt.

2.5.2 DM-Problematik

Während des gesamten Simulationsvorgangs werden zum Teil sehr große Datenmengen, die in verschiedenen Formaten vorliegen, aus unterschiedlichen Datenquellen verarbeitet. Dies stellt besondere Anforderungen an das Datenmanagement. Da im Simulationsworkflow selbst bisher keine expliziten Angaben über Quellen, Formate oder Datentransformationen

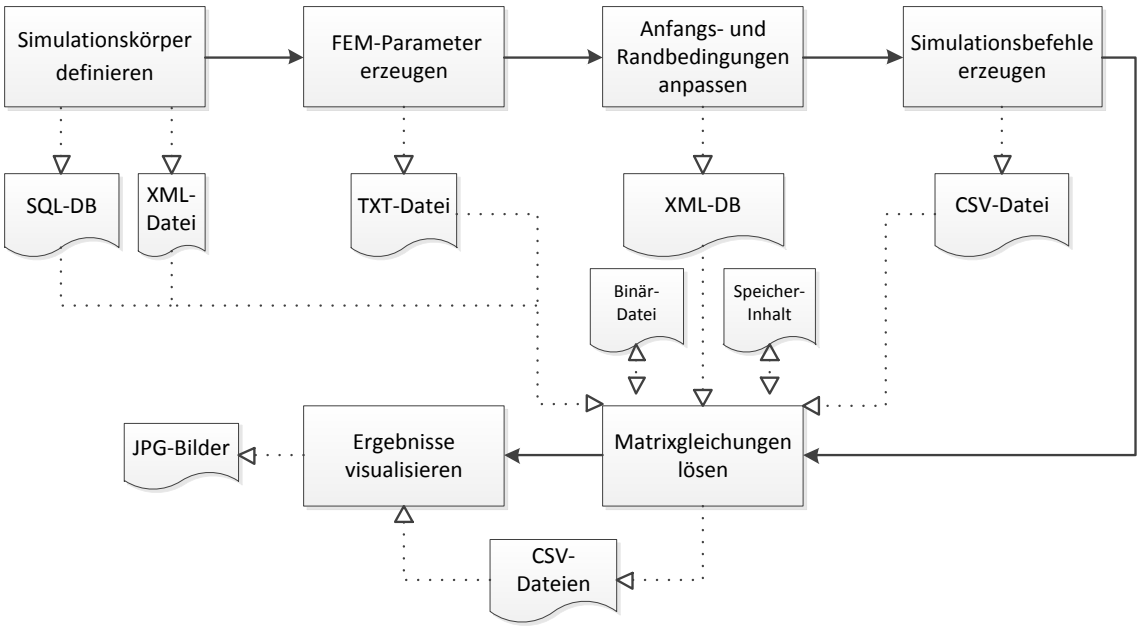

Abbildung 2.3: Beispiel für Datenquellen/-formate im PANDAS-Wf

gemacht werden, liegt es in der Verantwortung des Anwenders dafür zu sorgen, dass die einzelnen Simulationsprogramme für sie verwertbare Daten erhalten. Abbildung 2.3 zeigt zur Veranschaulichung eine fiktive Verteilung der PANDAS-Daten auf unterschiedliche Formate und Quellen, um die mögliche Verschiedenheit beteiligter Datenquellen zu verdeutlichen.

Aufgrund der vielen unterschiedlichen Datenformate, Quellsysteme und der enormen Datenmengen, die auftreten können, ist die manuelle Anpassung der Daten an die Programme in größeren Simulationsvorgängen kaum durchführbar. Für den Anwender entsteht dabei ein sehr hoher Zusatzaufwand, der mit zunehmender Komplexität auch zusätzliche Anforderungen an seine Qualifikation stellt. Selbst mit dem nötigen Fachwissen ist die eigenhändige Anpassung bei größeren Vorgängen äußerst fehleranfällig. Aus diesen Gründen ist es wünschenswert, die Verwaltung aller in einem Workflowkontext anfallenden Daten, Formate und Quellen sowie alle nötigen Datentransfers zwischen den Simulationsprogrammen automatisiert durchführen zu lassen.

Für die Verwaltung verschiedener Quellen und die Überführung in verschiedene Datenformate kann bereits SIMPL (s. Kapitel 3) verwendet werden. Die Aufgabe des Datentransfers muss mit Hilfe externer Dienste oder Workflow-Fragmente modelliert werden, welche für das PANDAS-Rahmenwerk bereits vorliegen. Die Einbindung dieser Dienste bzw. Workflow-Fragmente in den Workflow kann mit SIMPL realisiert werden, jedoch muss für jeden Datentransfer der richtige Dienst oder das richtige Workflow-Fragment per Hand ausgesucht und in den Workflow integriert werden. Ziel ist es nun, die benötigten Dienste bzw. Workflow-Fragmente anhand weniger Eingaben automatisch finden und in den Workflow integrieren zu lassen.

Das Hauptziel dieser Arbeit besteht darin zu untersuchen, ob und wie sich SIMPL unter Zuhilfenahme von Konzepten aus Champagne (s. Kapitel 4) erweitern lässt, um eine automatisierte Einbindung der benötigten Dienste bzw. Workflow-Fragmente zu erreichen.

3 SimTech - Information Management, Processes and Languages

In diesem Kapitel wird *SIMPL*, das Rahmenwerk zur Einbindung externer Datenquellen in ein sWfMS, behandelt. Zunächst werden Architektur und Funktionsweise vorgestellt, anschließend werden Möglichkeiten und Einschränkungen beim Einsatz des Rahmenwerks in einem sWfMS aufgezeigt. Die Ausführungen in diesem Kapitel basieren auf [RRS+11], sofern keine abweichenden Quellen angegeben werden.

Die meisten heutigen sWfMS bieten keine Möglichkeit zur integrierten Verwaltung externer Datenquellen. In einem Workflow müssen solche Daten bisher beispielsweise durch den gezielten Aufruf von Diensten, die auf die entsprechenden Datenquellen zugeschnittenen sind, verarbeitet werden (vgl. Abschnitt 2.3). Weiterhin erhöhen die Wahl geeigneter Datenquellen und die Implementierung benötigter Datentransformationen die Komplexität und den Zeitaufwand für den Entwickler. Daher ist es wünschenswert, innerhalb eines sWfMS eine Abstraktionsebene zu schaffen, die den Zugriff auf workflowexterne Datenquellen und die Verwaltung der darin enthaltenen Daten vereinheitlicht. Eine solche Abstraktionsebene würde dem Entwickler die Auseinandersetzung mit den Details bei der Verwaltung verschiedener Datenquellen ersparen. Von besonderem Vorteil wäre dies bei fachgebietsübergreifenden Simulationen mit unterschiedlichen Anforderungen innerhalb der einzelnen Fachgebiete.

Mit SIMPL wurde ein erweiterbares Rahmenwerk entworfen, um die fehlende Abstraktionsebene für die Datenverwaltung in einem sWFMS zu schaffen. Es stellt dem Entwickler vereinheitlichte Zugriffsmechanismen auf heterogene, externe Daten zur Verfügung. Die eigentlichen Zugriffsoperationen und Umwandlungen zwischen verschiedenen Datenquellen werden dabei von Metadaten innerhalb des SIMPL-Systems beschrieben. Dem Entwickler werden einige zusätzliche Workflowaktivitäten zur Verfügung gestellt, mit deren Hilfe sich die Datenverwaltung für beliebige Quellen innerhalb des Workflows modellieren lässt. Zusätzlich werden *DatenManagement-Patterns* zur Ausführung von DM-Operationen eingeführt, die die Modellierung von Simulationsworkflows weiterhin erleichtern sollen. Das SIMPL-Rahmenwerk soll dabei für beliebige Arten von Datenquellen und DM-Patterns erweiterbar sein.

Im Folgenden wird zunächst die Architektur und Funktionsweise von SIMPL sowie die Integration in ein sWfMS analog zu Abschnitt 2.3 beschrieben.

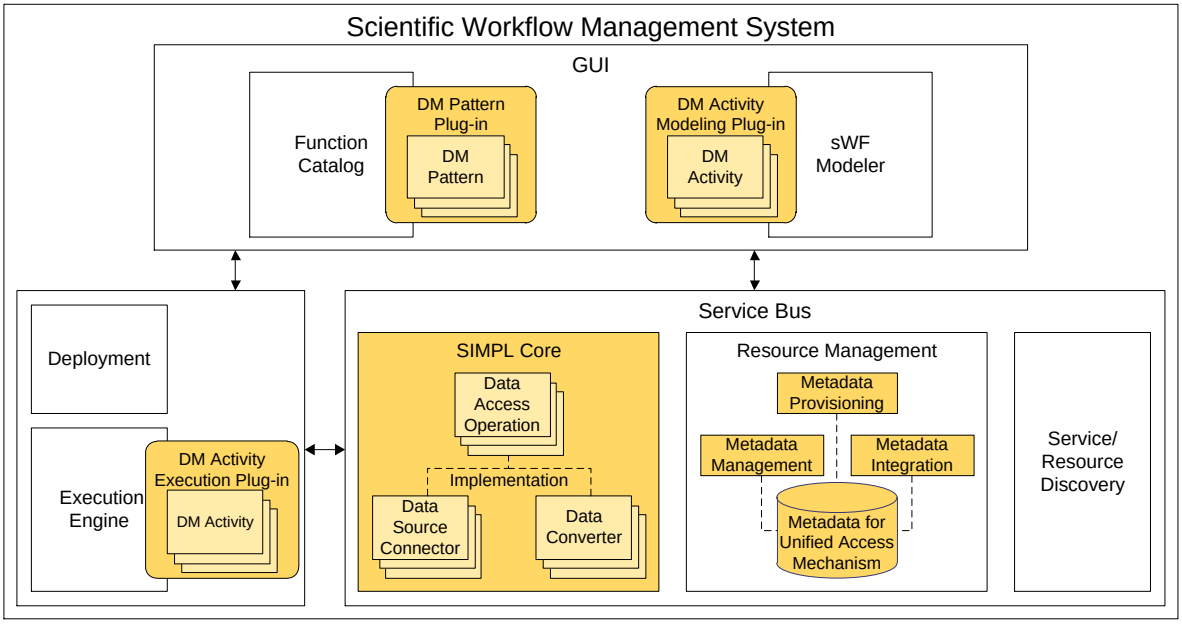

Abbildung 3.1: SIMPL-Rahmenwerk integriert in ein sWfMS (Vgl. [RRS+11])

3.1 Architektur

SIMPL erweitert die Sprache BPEL um Datenverwaltungsaktivitäten (*DM activities*). Diese Erweiterung erhielt die Bezeichnung BPEL-DM (Business Process Execution Language extension for Data Management). Die neu definierten Aktivitäten werden zur nahtlosen Anbindung von Datenquellen in Workflowprozesse eingesetzt. Sie rufen eine SIMPL-Laufzeitkomponente auf, welche die gewünschten Operationen auf den entsprechenden Datenquellen ausführt. Die Anbidung der Datenquellen erfolgt dabei mit Hilfe von *Data Source Connectors*, die die Schnittstellen zu den verschiedenen Quellsystemen bilden. Für jede anzubindende Datenquelle muss dabei ein eigener Data Source Connector zur Verfügung gestellt werden, der den Zugriff auf die Quelle implementiert.

Abbildung 3.1 zeigt auf Grundlage von Abbildung 2.1 (s. Abschnitt 2.3) die durch SIMPL realisierte Erweiterung eines sWfMS um eine Abstraktionsebene zum Zugriff auf und zur Verwaltung von Daten. Die im Vergleich zu Abbildung 2.1 fehlenden Komponenten sind für die Erweiterung nicht relevant und wurden der Überschaubarkeit halber weg gelassen. Im Folgenden werden die Komponenten von SIMPL beschrieben.

Die **SIMPL Core**-Komponente innerhalb des Service Bus liefert vereinheitlichte logische Schnittstellen zu Datenquellen jeglicher Art. Die Implementierung dieser logischen Schnittstellen für konkrete Datenquellen übernehmen dabei die Data Source Connectors zusammen mit *Data Converters*, die die notwendigen Schematarnsformationen durchführen.

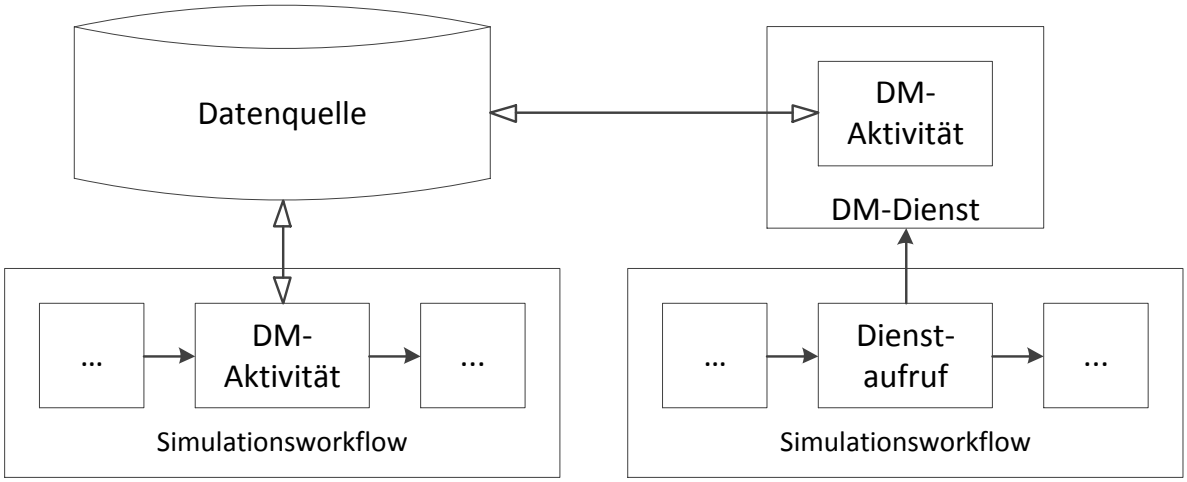

Abbildung 3.2: Links: Verwendung einer SIMPL DM-Aktivität innerhalb eines Workflows Rechts: Aufruf eines gekapselten Dienstes, der eine DM-Aktivität beinhaltet.

Die **Resource Management**-Komponente des sWfMS wird um SIMPL-spezifische Metadaten erweitert. Sie enthalten u.a. Informationen über die Zugriffsmechanismen für die unterstützten Datenquellen. Das Resource Management ist als relationale Datenbank implementiert.

Das **DM Activity Modeling Plug-In** erweitert den sWF Modeler um die durch BPEL-DM spezifizierten Aktivitäten. Der Entwickler kann diese Aktivitäten verwenden, um einen Workflow mit Hilfe der SIMPL-Erweiterung zu modellieren.

Das **DM Activity Execution Plug-In** sorgt für die Ausführung der neuen Aktivitäten zur Laufzeit. Dabei gibt es grundsätzlich zwei Arten, diese Aktivitäten zu verwenden: Sie können entweder direkt in den Simulationsworkflow integriert werden oder in Form eines gekapselten, eigenständigen Workflows, der als Dienst zur Verfügung gestellt wird, aus dem eigentlichen Simulationsworkflow heraus aufgerufen werden (vgl. Abbildung 3.2).

Der **Function Catalog** des sWfMS wird um ein *DM-Pattern Plug-In* erweitert, welches den Entwickler beim Modellieren der benötigten DM-Operationen unterstützt. Anstatt jede Datenoperation von Hand einzugeben, kann er aus Vorlagen wählen, die komplexe Operationen unter Angabe weniger Parameter realisieren. Diese Vorlagen müssen vor der Ausführung des Workflows in ausführbare Operationen umgewandelt werden.

3.2 DM-Aktivitäten

Nachfolgend werden die wichtigsten Aktivitäten von BPEL-DM erläutert. Jede DM-Aktivität besitzt eine spezielle BPEL-Variable (*data source reference variable*), die auf die Datenquelle verweist, auf die sie zugreift. Eine Datenquelle verwaltet *Datencontainer*, die jeweils bestimmte und identifizierbare Datenobjekte repräsentieren, beispielsweise eine Tabelle in

einer Datenbank oder eine Datei in einem Dateisystem. Sie werden in einem Workflow durch *data container reference variables* referenziert und können anhand dieser eindeutig identifiziert und über das Resource Management der korrekten Datenquelle zugeordnet werden. Mit Hilfe von *data set variables* können Daten aus externen Quellen zur weiteren Verarbeitung in den Workflowkontext geladen werden. Die Inhalte dieser Variablen sind durch XML-Schemadefinitionen festgelegt und müssen daher im XML-Format vorliegen. Für tabellenbasierte Daten wird beispielsweise die RowSet-Struktur, die einen XML-basierten Datentyp für Tabellendaten definiert, verwendet. Die Schemadefinitionen müssen dabei die Besonderheiten unterschiedlicher Datenquellen berücksichtigen. Die wichtigsten DM-Aktivitäten, die diese Variablen verwenden, werden im Folgenden beschrieben.

Mit **IssueCommand** können Befehle an Datenquellen übermittelt werden, mit deren Hilfe Daten direkt manipuliert oder definiert werden können. Diese Aktivität beinhaltet neben der *data source reference variable* den auszuführenden Befehl für die jeweilige Datenquelle.

Mit **RetrieveData** werden Daten aus einer Datenquelle ausgelesen und zur weiteren Verarbeitung in den Workflowkontext geladen. Diese Aktivität enthält die *data source reference variable*, den quellenabhängigen Abfragebefehl sowie eine *data set variable*, in der nach erfolgreicher Ausführung die Ergebnisdatensätze abgespeichert werden.

Mit **WriteDataBack** werden Daten aus dem Workflowkontext in eine Datenquelle geschrieben. Dabei werden Datensätze aus einer *data set variable*, die etwa zuvor mit Hilfe von *RetrieveData* belegt wurde, in einen durch eine *data container reference variable* spezifizierten Datencontainer eingefügt.

Innnerhalb eines DM-Befehls können Workflowvariablen als Platzhalter verwendet werden, indem sie beispielsweise von Hashmarks („#“) umschlossen werden. Die Ausführungsengine erkennt diese Syntax und löst die eingebetteten Variablennamen vor dem Abschicken des Befehls an die Datenquelle nach ihrem Inhalt auf. Auf diese Weise können z.B. *data container reference variables* in der FROM-Klausel eines SQL Befehls angegeben oder andere BPEL-Variablen für Vergleichsoperationen innerhalb eines DM-Befehls herangezogen werden.

Die durch das *DM Activity Execution Plug-In* erweiterte Ausführungsengine des sWfMS erwartet bei jeder dieser Aktivitäten eine Rückmeldung über den Erfolg oder Misserfolg des jeweiligen Befehls von der Datenquelle und unterbricht im Fehlerfall die Ausführung des Workflows, um eine Fehlerbehandlung bzw. -kompensation durchzuführen.

3.3 DM-Patterns

Mit der Erweiterung des Funktionskatalogs um Vorlagen für DM-Operationen wird eine höhere Abstraktionsstufe für die Verwaltung der Daten geboten. Der Entwickler kann Datenoperationen, für die ein Pattern vorliegt, modellieren, ohne die datenquellenspezifischen Befehle selbst formulieren zu müssen. Beispielsweise soll ein JOIN zweier Datensätze mit Hilfe eines Patterns durch die Angabe zweier Eingabevariablen, einer Ausgabevariablen sowie der *join condition* realisiert werden. Welche Aktivitäten durch das Pattern im einzelnen ausgeführt werden hängt schließlich vom Inhalt der Parameter ab. Handelt es sich bei allen

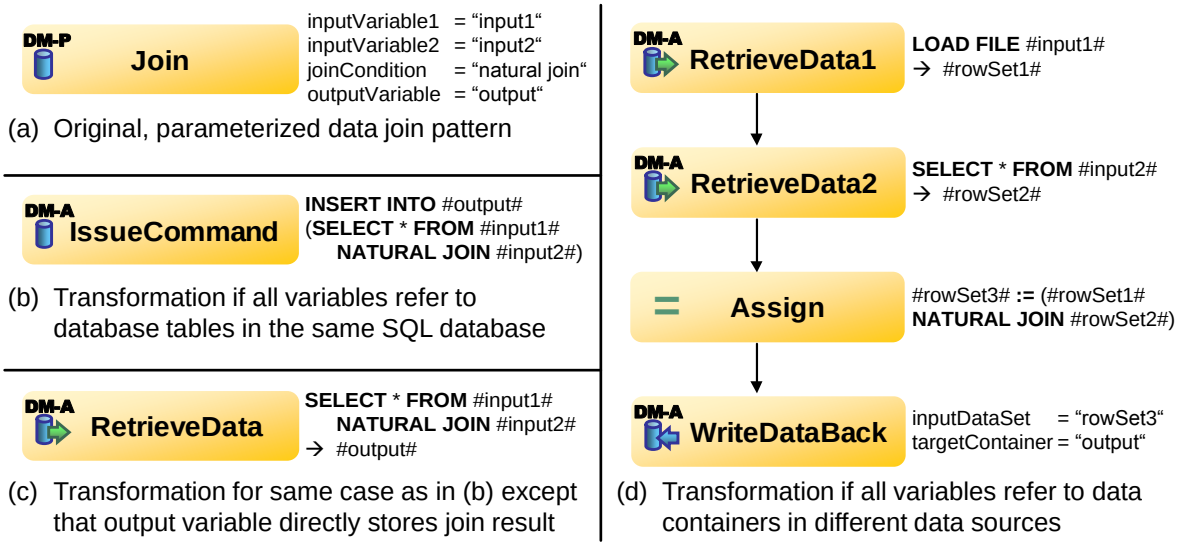

Abbildung 3.3: Join-Pattern und seine ausführbaren Umformungen gemäß [RRS+11]

Variablen um *data container reference variables* auf Tabellen der selben SQL-Datenbank, so kann das Pattern eine *IssueCommand*-Aktivität verwenden, die den entsprechenden SQL-Befehl an die Datenbank sendet.

Verwendete DM-Patterns werden vor der Ausführung anhand von *Umformungsregeln* abhängig von den Eingabeparametern zu Workflowteilen umgeschrieben, welche schließlich genau die Aktivitäten beinhalten, die die ursprünglich gewünschte Operation realisieren. Abbildung 3.3 zeigt verschiedene Umformungen eines *Join*-Patterns. Teil (a) zeigt dabei die Ursprünglich Form, wie sie vom Entwickler vewendet wird. Die Teile (b) bis (d) zeigen die abhängig von den Eingabeparametern verschiedenen Umformungen dieses Patterns.

Die Anwendung der Umformungsregeln kann erst dann erfolgen, wenn alle an der Operation des DM-Pattern beteiligten Datencontainer bzw. -quellen feststehen. Bei der statischen Anbindung von Datenquellen vor der Ausführung des Workflows genügt es, die Unformungsregeln kurz vor dem Deployment anzuwenden. Sollen Datenquellen zur Laufzeit angebunden werden, können die Patterns in Workflow-Fragmente umgewandelt und mit Hilfe von *Prozessfragmentierungstechnologien* eingebunden werden [EUL09].

DM-Patterns sind bisher nur im Ansatz beschrieben und sollen nun konkretisiert werden. Eines der wichtigsten Patterns ist dabei das Datentransfer-Pattern, welches den Transfer von Daten einer Quelle zu einer anderen außerhalb des SIMPL-Core realisieren soll. Durch ein solches Pattern wird die in Abschnitt 2.5.2 eingeführte Problematik gelöst, ohne dass die betroffenen Daten zuvor in den Workflowkontext geladen werden müssen. Um Ideen zur Umsetzung eines solchen DM-Patterns zu bekommen, werden im nächsten Kapitel Architektur und Konzepte eines Propagationsmanagers untersucht, der ähnliche Datenoperationen realisiert.

4 Champagne

Ein Unternehmen umfasst heutzutage oft mehrere eigenständige Informationssysteme. Es werden Daten von der untersten (operativen) Ausführungsebene bis hin zur obersten Planungsebene verwaltet. Dabei werden oft mehrere EDV-Systeme mit proprietären Verwaltungs- und Zugriffsmechanismen eingesetzt. Diese Systeme bilden zusammen ein heterogenes Umfeld, in dem oft die selbe Information in den verschiedenen Teilsystemen durch unterschiedliche Daten repräsentiert wird. Diese können in verschiedenen Quellen, Formaten und Strukturen vorliegen. Die verschiedenen Repräsentationen einer Information müssen bei Änderungen in einem Teilystem in allen anderen Systemen angepasst werden. In diesem Kapitel wird das System *Champagne* (**cha**nge pro**pa**gation **m**anager) vorgestellt, das die Konsistenz durch *Datenänderungspropagation* realisiert. Zunächst wird die Konsistenzproblematik näher untersucht. Anschließend werden Funktionsweise und Architektur von Champage vorgestellt. Die Ausführungen in diesem Kapitel basieren auf [Hei11], sofern keine abweichenden Quellen angegeben werden.

4.1 Anomalien bei Datenänderungen

Beim Ändern von Daten in heterogenen Systemen können, ähnlich wie bei nicht normalisierten Tabellen einer relationaler Datenbank, Änderungsanomalien auftreten. Abbildung 4.1 zeigt ein einfaches Beispiel für die Repräsentation der selben Information in verschiedenen DM-Systemen eines Unternehmens und die daraus resultierende Anomalie beim Auftreten einer Änderung. Beim Anlegen eines Kundenkontos für den Login-Bereich einer Webseite wird neben den Kundendaten ebenfalls ein Konto für die Diskussionsforen der Webseite erstellt. Die zwei Systeme können dabei heterogen sein. Werden nun die Kundendaten verändert (entweder durch den Kunden selbst oder durch einen Systemadministrator), wird diese Änderung vom unabhängigen DM-System der Diskussionsforen nicht registriert und es enthält zunächst veraltete bzw. inkorrekte Daten. Um die Konsistenz zwischen den zwei Systemen wiederherzustellen, muss in einem weiteren Schritt die entsprechende Änderung innerhalb des Forensystems vorgenommen werden.

Um die von der Änderungsanomalie betroffenen Daten identifizieren zu können, muss zunächst ein Weg gefunden werden, bestehende Abhängigkeiten zwischen verschiedenen Daten zu beschreiben. Anschließend müssen anhand dieser Abhängigkeitsbeschreibungen auftretende Änderungen an die betroffenen DM-Systeme propagiert werden. Zur Automatiserung solcher Änderungspropagationen wurde Champagne entwickelt. Ursprung und momentanes Hauptanwendungsgebiet ist dabei das Produktionsumfeld, wobei oft mehrere

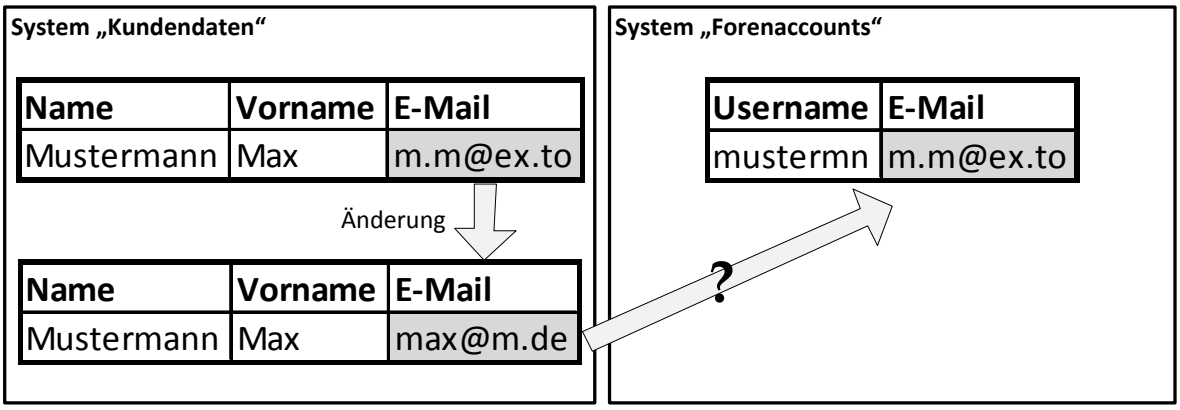

Abbildung 4.1: Anomalie bei Änderungen mehrfach repräsentierter Information

Hallen, Maschinen oder andere unabhängige Ressourcen in unterschiedlichen DM-Systemen verwaltet werden müssen.

4.2 Basiskonzepte

Champagne bietet Entwurfszeitkomponenten zum Modellieren von Regeln, die Abhängigkeiten zwischen Daten beschreiben, und Skripten, die die Umwandlungen zwischen den an einer Abhängigkeit beteiligten Schemas umsetzen. Die Technologie orientiert sich dabei an etablierten Konzepten aus dem Bereich der Replikation von (homogenen) Datenbanken.

Grundlage der Änderungspropagation in Champagne sind *Geschäftsobjekte* (GO), die aus mehreren Daten oder weiteren Geschäftsobjekten zusammengetzt sein können. Beipielsweise wäre ein GO vom Typ *Adresse* aus den Daten Straße, Postleitzahl und Ort zusammengesetzt. Ein GO-Typ *Adresse* kann selbst wiederum Teil eines GO-Typs *Kunde* sein.

4.2.1 Abhängigketen und Änderungsbeschreibungen

Eine *Abhängigkeit* ist eine gerichtete Beziehung zwischen einem GO-Typ aus einem Quellsystem und einem oder mehreren GO-Typen in (verschiedenen) Zielsystemen. Damit Änderungen an Geschäftsobjekten an andere Systeme weitergeleitet werden können, werden diese nach ihrer Erfassung in Form von *Änderungsbeschreibungen* festgehalten. Ändern sich ein oder mehrere Attributte eines Geschäftsobjekts, so ändert sich dessen Zustand. Der Zustand eines Geschäftsobjekts ist die Menge aller seiner Attributwerte. Wie bereits im vorigen Abschnitt beschrieben, kann dabei ein Attribut ein weiters Geschäftsobjekt sein.

Es gibt drei grundsäzliche Arten, wie sich der Zustand eines Geschäftsobjekts verändern kann:

create : Ein neues Geschäftsobjekt wird erzeugt.

update : Ein oder mehrere Attribute eines bestehenden Geschäftsobjekts werden verändert.

delete : Ein bestehendes Geschäftsobjekt wird gelöscht.

Eine *Änderungsbeschreibung* ist ein Objekt, das den Vorgang der Änderung eines Geschäftsobjekts beschreibt. Sie wird durch den folgenden 6-Tupel beschrieben:

$$AB = (S, GT, A, B, D, TS)$$

S : System in dem die Änderung auftrat bzw. angewedet werden soll.

GT : GO-Typ des betroffenen Geschäftsobjekts.

A : Die Art der Änderung (create, update oder delete).

B : Zustandsbeschreibung des Geschäftsobjekts vor der Änderung (*Davor-Zustand*).

D : Zustandsbeschreibung des Geschäftsobjekts nach der Änderung (*Danach-Zustand*).

TS : Zeitstempel der Änderung.

Eine Änderungsbeschreibung liegt in der Regel im XML-Format vor. Zur Verbesserung des Laufzeitverhaltens von Propagationen können diese in eine interne Repsäsentation, z.B. DOM (*Document Object Model*) [dom], umgewandelt werden. Bei der Verarbeitung einer Propagation können sich alle Elemente des Tupels ändern. Aus dem Quellsystem (S) wird das Zielsystem. Der GO-Typ ändert sich, da in verschiedenen Systemen unterschiedliche Bezeichner für die GO-Typen existieren, auch wenn sie die selben Datenstrukturen repeäsentieren. Die Änderungsart A kann sich ebenfalls ändern, wenn z.B. ein Geschäftobjekt in einem System hinzugefügt wird (create), während ein anderes System die Gesamtzahl der sich im System befindlichen Geschäftsobjekte dieses GO-Typs speichert. In diesem Fall muss die create-Aktion für das Zielsystem in eine update-Aktion umgewandelt werden. Die Zustände B und D sind abhängig von A definiert: Für A=*create* ist B=*NULL*, für A=delete ist D=*NULL*. Für A=*update* sind sowohl B als auch D mit Zuständen belegt.

4.2.2 Propagationsskripte

Findet eine Datenänderung in einem durch *Champagne* erweiterten System statt, so wird diese durch einen *Adapter* (siehe Abschnitt 4.3.4) erkannt und es wird eine Änderungbeschreibung erzeugt. Diese muss nun für jedes Ziel einer modellierten Abhängigkeit in jeweils eine Änderungsbeschreibung für die betroffenen Zielgeschäftsobjekte umgewandelt werden. Dazu werden *Propagationsskripte* ausgeführt, welche die für eine Propagation notwendigen Schritte beschreiben. Propagationsskripte benitzen eine *input declaration*, anhand derer die Änderungsbeschreibungen, die sie nicht betreffen, gefiltert werden können. Ein Propagationsskript besteht aus drei Teilen:

- Ein Input-Teil legt zunächst fest, in welchem Geschäftobjekt und welcher Datenquelle die entsprechende Änderung auftrat.
- Ein Verarbeitungsteil, in dem die Änderungsbeschreibung in die für die Zielsysteme korrekte Form gebracht wird.
- Ein Output-Teil definiert anschließend alle von der Abhängigkeit betroffenen Zielgeschäftsobjekte und Zielsysteme.

Im Verarbeitungsteil werden dabei ein oder mehrere *Transformationsskripte* verwendet, die durch das Propagationsskript koordiniert werden und die Änderungsbeschreibung aus dem Schema des Quellystems einer Abhängigkeit in Änderungsbeschreibungen im Schema der Zielsysteme umwandeln.

Für die Repräsentation der Propagationsskripte wurde die *XML Propagation Definition Language* (XPDL[1]), einer Erweiterung der Workflow-Sprache XRL (*Extensible Routing Language*) [AVK01, VHA02, AK03] , entwickelt.

Im Folgenden wird die Architektur von Champagne näher untersucht.

4.3 Architektur

Die drei Hauptkomponenten eines (allgemeinen) Propagationssystems bilden ein *Abhängigkeitsmanager*, ein *Propagationsmanager* und ein *Repository* [CHRM01]. Mit Hilfe des Abhängkeitsmanagers werden während der Entwurfsphase die Abhängigkeiten zwischen den Daten in Form von Propagationsskripten erstellt bzw. modifiziert. Der Propagationsmanager ist für die Áusführung dieser Propagationsskripte zur Laufzeit zuständig. Beide Kompentenen greifen dabei auf eine interne Datenbasis (Repository) zurück, in der alle systembezogenen Daten verwaltet werden. Abbildung 4.2 zeigt eine Übersicht der Komponenten von Champagne.

4.3.1 Repository

Das Repository ist der zentrale Datenspeicher des Champagne-Systems. Es wird durch ein relationales Datenbankmanagementsystem (*RDBMS*) implementiert. Die darin enthaltenen Daten werden ausschließlich zur Entwurfszeit durch Komponenten des Abhängigkeitsmanagers erstellt bzw. verändert.

Im Einzelnen werden hier folgende Daten verwaltet:

- Die **Propagationsskripte**, die Abhängigkeiten und Propagationsprozesse beschreiben.
- Die **Transformationsskripte**, die von den Propagationsskripten zur Umwandlung von Zustandsbeschreibungen verwendet werden.

[1]Diese Abkürzung wird ungünstigerweise bereits durch die *XML Process Definition Language* verwendet

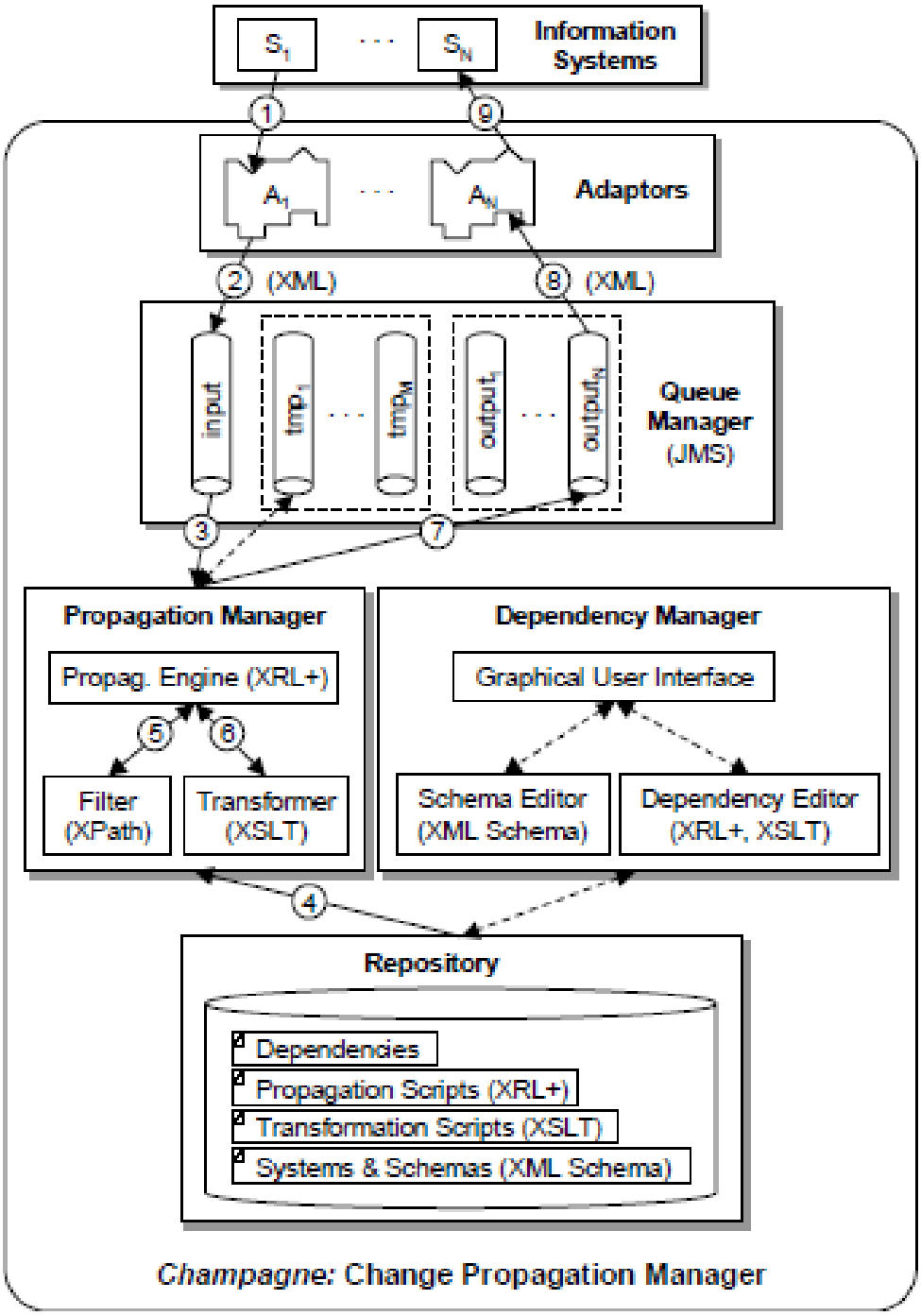

Abbildung 4.2: Architektur von Champagne. [RCHM02]

- **Schemainformationen** zu den Änderungsbeschreibungen von Geschäftsobjekten.
- **Metadaten** über die angebundenen Informationssysteme, wie deren Bezeichnungen und Zugriffsarten.

  Die Transformationsskripte liegen in *XSLT* bzw *XQuery* vor und wandeln die XML-basierten Änderungsbeschreibungen, die von einem Quellobjekt ausgehen, in das Format der Zielobjekte um.

4.3.2 Abhängigkeitsmanager (Dependency Manager)

Der Abhängigkeitsmanager unterstützt den Entwickler bei der Erstellung und Bearbeitung von Abhängigkeiten und Propagationsskripten. Mit Hilfe eines *Abhängigkeitseditors* kann er die benötigten Propagationsskripte und die von ihnen verwendeten Dokumente erstellen. Dazu gehören Systeminformationen über angebundene Datenquellen, Schemas und Transformationsskripte. Systeminformationen werden durch einen einfachen Dialog eingegeben. Schemas und Transformationsskripte liegen in standard XML-Technlogie vor (z.B. XSLT für Transformationsskripte). Zum Entwurf und zur Bearbeitung dieser können daher externe XML-Standardwerkzeuge verwendet werden, die vom Abhängigkeitsmanager eingebunden und bei Bedarf aufgerufen werden können. Durch sie wird der in Abbildung 4.2 dargestellte Schema-Editor realisiert. Die Kommunikation mit den Standardwerkzeugen findet außerhalb des Repositorys über das Dateisystem statt, indem die fertigen Dokumente von den Werkeugen in Dateien abgespeichert und anschließend vom Abhängigkeitsmanager in das *Repository* transferiert werden.

Die Propagationsskripte selbst können nicht mit Hilfe von Standardwerkzeugen erstellt oder bearbeitet werden, da sie spezielle Ausdrücke enthalten, die durch die *XML Propagation Lagnuage* (XPDL) interpretiert werden. Sie werden daher ausschließlich durch den Abhängigkeitsmanager erstellt und verändert. Die Erweiterung von XPath ist notwendig, da bei der Ausführung der Propagationsskripte unter anderem ein Zugriff auf den Davor-Zustand einer Änderungsbeschreibung benötigt wird, wobei im Fall der Nichtexistenz (z.B. bei *CREATE*) des Davor-Zustands der Danach-Zustand zurückgegeben wird.

4.3.3 Propagationsmanager (Propagation Manager)

Der Propagationsmanager bildet die Kernkomponente und Laufzeitumgebung des Systems. Sobald eine Änderung festgestellt wird, führt ein *Prozessmanager* die Propagationsskripte auf einer Ausführungsengine gemäß der in dem Repository abgespeicherten Informationen aus. Die Änderungsnachrichten werden dabei über eine Komponente zum Warteschlangenzugriff (*Queue-Access*) aus der Eingangswarteschlange (*input*) des *Queue Managers* entnommen. Über eine Komponente für den Zugriff auf das Repository (*Repository Access*) werden anschließend die benötigten Propagationsskripte durch einen *Prozessmanager* gestartet. Die beiden Zugriffskomponenten kapseln die Funktionalität der dahinterliegenden Systeme.

Eine Ausführungsinstanz eines Propagationsskriptes ist ein *Propagationsprozess*. Propagationsprozesse erzeugen die Änderungsanforderungen und übergeben sie als Nachrichten in eine Ausgangswarteschlange (*output*). Die Warteschlangen werden von einem Warteschlangenmanager (*Queue Manager*) verwaltet.

4.3.4 Adapter (Adaptors)

Adapter bilden die Schnittstellen von Champagne zu den verschiedenen Informationssystemen. Dazu muss für jede Datenquelle ein passender Adapter bereitgestellt werden. *Quelladapter* sind für das Erkennen von Datenänderungen innerhalb eines Informationssystems sowie für die korrekte Repräsentation und Weiterleitung dieser Änderungen an das Propagationssystem in Form von Änderungsbeschreibungen zuständig. *Zieladapter* erhalten Änderungsbeschreibungen und müssen diese in entsprechende Befehle für die betroffene Datenquelle umwandeln. Durch den in der Änderungsbeschreibung enthaltenen Davor-Zustand kann der betroffene Datensatz eines Informationssystems identifiziert und anschließend mit dem Inhalt des Danach-Zustandes überschrieben werden.

Adapter müssen außerdem Mechanismen zur Fehlerbehandlung bzw. Konfliktbeseitigung bereitstellen. Insbesondere Zieladapter können beim Senden von Änderungsbefehlen auf Fehlermeldungen seitens des Informationssystems stoßen, beispielsweise wenn die vom Propagationsprozess erzeugte Änderungsbeschreibung gegen bestehende Konsistenzregeln des Zielsystems verstößt. Die Auflösung derartiger Konflikte geschieht mit Hilfe einer Benutzerschnittstelle (*GUI*) des Adapters zur Laufzeit. Quell- und Zieladapter werden oft als Einheit implementiert, da angeschlossene Informationssysteme meist sowohl Quell- als auch Zielsysteme von Änderungspropagationen sind.

5 Kombination der Technologien

In diesem Kapitel wird untersucht, ob die in Abschnitt 2.5.2 beschriebene Problemstellung des Datentransfers in Simulationsworkflows mit SIMPL (Kapitel 3) und mit Hilfe von Konzepten bzw. Komponenten aus Champagne (Kapitel 4) gelöst werden kann. Zunächst werden Gemeinsamkeiten und Unterschiede der beiden Systeme aufgezeigt. Danach werden verschiedene Ansätze zur Einbettung von Champagne bzw. dessen Komponenten innerhalb von SIMPL diskutiert. Schließlich wird untersucht, ob sich Champagne im Gegenzug mit SIMPL kombinieren lässt, so dass dadurch evtl. Probleme bei der Datenänderungspropagation gelöst werden können.

5.1 Gemeinsamkeiten und Unterschiede

Die beiden vorgestellten Systeme SIMPL und Champagne sind beide für die Handhabung von multiplen, heterogenen Datenquellen konzipiert. Bei SIMPL handelt es sich um eine Erweiterung der Architektur eines sWfMS, die den Zugriff auf diese Datenquellen von innerhalb eines Workflows ermöglicht und dem Entwickler Werkzeuge für die Manipulation, das Auslesen oder Zurückschreiben der darin enthaltenen Daten in Form von Workflow-Aktivitäten zur Verfügung stellt. Im Gegensatz dazu ist Champagne weder zur Anbindung von Datenquellen an übergeordnete Systeme oder Workflows, noch zur selektiven Manipulation von Daten vorgesehen. Stattdessen handelt es sich um ein eigenständiges, unabhängiges System, das Datenquellen überwacht und Datenänderungen möglichst zeitnah und unscheinbar an andere Datenquellen propagiert. Es läuft dabei parallel zu den Systemen bzw. Workflows, die für die eigentliche Datenverwaltung zuständig sind. Dieser markante Unterschied in der Funktionsweise spiegelt sich auch in der unterschiedlichen Architektur der beiden Systeme wider, was die Kombination erschwert. Im Folgenden werden einzelne Komponenten der beiden Systeme gegenübergestellt und miteinander verglichen.

5.1.1 Data Source Connector, Data Converter und Adapter

Beide Systeme kommunizieren mit verschiedenen Datenquellen und manipulieren darin enthaltene Daten.

Die Umwandlung der internen Repräsentation in Befehle für die ausführende Datenquelle übernehmen die Komponenten *Data Source Connector* (SIMPL) bzw. *Adapter* (Champagne). In beiden Fällen muss zur Anbindung einer Datenquelle ein entsprechender Connector bzw. Adapter vorhanden sein. Bei SIMPL übernehmen zusätzliche *Data Converters* die Anpassung

der Daten an die Schemas der Verschiedenen DM-Systeme. Die beiden Systeme sind durch dieses Konzept um beliebige Datenquellen erweiterbar.

Eine naheliegende Überlegung ist die Frage, ob sich ein Adapter bzw. Connector im jeweils anderen System einsetzen und somit Implementierungsaufwand einsparen lässt. Bereits auf dem ersten Blick scheinen jedoch beide nicht ohne Erweiterungen für den Einsatz im jeweils anderen System geeignet zu sein:

- SIMPL-Connectors bzw. Data Converters besitzen keine Logik zur Handhabung von Geschäftsobjekten oder zur Umwandlung von Änderungsbeschreibungen in DM-Befehle und bietet beispielsweise auch keine eigene Schnittstelle (GUI) zur Konfliktbeseitigung. Letztere wird auf der SIMPL-Seite durch das WfMS zur Verfügung gestellt und muss in der Regel bereits in den Workflow-Modellen zur Entwurfszeit festgelegt werden. Schreiboperationen beziehen sich bei SIMPL immer auf eine bestimmte Datenmenge, die in eine Datenquelle geschrieben wird, während bei Champagne die tatsächlich zu schreibenden Daten erst vom Adapter aus der Änderungsbeschreibung generiert werden müssen. Data Source Connectors und Data Converters könnten dennoch als Teile von Adaptern verwendet werden. Diese könnten bei der Implementierung neuer Adapter Aufwand einsparen.
- Champagne-Adapter können in SIMPL ebenfalls nicht ohne weiteres eingesetzt werden. Obwohl sie bei einer schreibenden DM-Operation aufgrund ihrer Unterstützung von Änderungsbeschreibungen und Geschäftsobjekten zunächst mächtiger als ihre SIMPL-Pendants erscheinen, so bieten sie offensichtlich keinerlei Unterstützung für Lesebefehle. Sie sind ingesamt für den Einsatzzweck in SIMPL überdimensioniert und würden die Effizienz des Systems herabsenken. Von Quelladaptern werden lediglich Datenänderungen erfasst und in Form von Änderungsbeschreibungen weitergeleitet. Für den Einsatz in SIMPL fehlt hier also ein datenlesender Bestandteil.

5.1.2 Resource Management und Repository

Sowohl SIMPL als auch Champagne verwenden eine eigene Datenbasis zur Verwaltung von Metainformationen, die vom System benötigt werden. In beiden Fällen wird diese Datebasis durch jeweils ein RDBMS realisiert, so dass bei gleichzeitigem bzw. kombiniertem Einsatz beider Systeme ein einzelnes RDBMS ausreichen würde. Hier wäre zu untersuchen, ob und wie Metainformationen, etwa über Datenquellen oder Schemas, auch vom jeweils anderen System sinnvoll mitverwendet werden könnten.

5.1.3 DM Activity Modeling Plug-In und Abhängigkeitsmanager

Die primäre Benutzerschnittstelle zur Entwicklung bildet in SIMPL der *sWF Modeler* mit dem DM Activity Modeling Plug-In sowie der *Function Catalog* mit den DM-Patterns. Bei Champagne werden mit Hilfe des eigenständigen Abhängigkeitsmanagers alle vom System benötigten Daten durch den Benutzer eingegeben. Eine Abstraktionsunterstützung, wie in

Form der DM-Patterns, ist dabei nicht vorgesehen. Das Grundkonzept von SIMPL definiert hingegen keine Benutzerschnittstelle zur Manipulation der Daten innerhalb des Resource Management. Bei der Implementierung von SIMPL wird dafür ein Web-Interface zur Verfügung gestellt.

Da die Propagationsskipte von Champagne in einer Erweiterung der Workflow-Sprache XRL implementiert werden, könnten die sWf-Komponenten zur Erstellung dieser Skripte herangezogen werden. In Abschnitt 5.5 wird dazu der Einsatz von Workflows und SIMPL bei der Datenpropagation untersucht.

5.1.4 Laufzeitkomponenten

Die Laufzeitkomponenten realisieren die jeweiligen (Haupt-)Funktionen ihrer Systeme. Das DM Activity Execution Plug-In der Ausführungsengine, zusammen mit dem SIMPL Core, und der Propagationsmanager von Champagne bilden jeweils die Haupt-Laufzeitkomponenten ihres Systems. Der Propagationsmanager ist für die korrekte Ausführung der Propagationsskripte und allen damit verbundenen Transformationen zuständig. Das DM Activity Execution Plug-In implementiert die DM-Aktivitäten, die durch das Modeling Plug-In bereitgestellt werden. Die Ausführungsengine implementiert den Rest der Workflow-Ausführung.

5.2 Champagne als Erweiterung von SIMPL

In Abschnitt 3.3 wurde das Konzept der DM-Patterns vorgestellt. Es wurde ein Datentransfer-Pattern vorgeschlagen, mit dessen Hilfe der Transfer von Daten einer oder mehrerer Quellen zu einem ohne mehrerer Ziele ohne den Umweg über den Workflow bewerkstelligt werden soll. Es soll nun untersucht werden, welche Konzepte bzw Komponenten aus Champagne dabei helfen können, ein solches Datentransfer-Pattern in SIMPL zu realisieren.

Das Datentransfer-Pattern soll an einer bestimmten Stelle innerhalb eines Workflows eingesetzt und aufgerufen werden. Das bedeutet konkret, dass das System anhand von Eingabeparametern, welche die Quell- und Zielsysteme enthalten, nach Möglichkeit selbstständig das richtige Workflow-Fragment einbinden oder den richtigen Dienst aufrufen soll. Dazu muss eine Zuordnung zwischen den Eingabeparametern und den exitierenden Diensten bzw. Workflow-Fragmenten hergestellt werden. In Champagne werden Propagationsskripte inerhalb des Propagationsmanagers ausgeführt und mit Hilfe ihrer *input declaration* und Änderungsbeschreibungen der Geschäftsobjekte vor ihrer Ausführung gefiltert. Die Anpassung des Propagationsmanagers an die Anforderungen des Datentransfer-Patterns und die anschließende Verwendung der Champagne-Komponente zur Realisierung wären sehr umständlich.

Das Champagne-System ist für die kontinuierliche Überwachung und Propagation von Datenänderungen auf verschiedenen Datenquellen konzipiert. Ein Aufruf bzw. die Einbidung von Champagne als isolierter Ausführungsschritt in einem Workflow ist nicht vorgesehen

und wird durch dessen Komponenten in der jetzigen Konzeption auch nicht begünstigt. Die Hauptfunktionalität von Champagne, in Form der Überwachung und Propagation von Datenänderungen, wird wiederum für SIMPL bzw. das Datentransfer-Pattern nicht benötigt, da dort komplette Datensätze an vorher definierten Stellen innerhalb eines Workflows von Quelle A zu Quelle B transferiert werden sollen. Eine weitere wichtige Funktion von Champagne, nämlich die Synchronisation der einzelnen Propagationsskripte, findet im Umfeld eines Workflows und insbesondere des Datentransfer-Patterns keine Anwendung, da hier alle Abläufe einem vorbestimmten, vom Workflow-Modell definierten Schema untergeordnet sind.

Champagne ist in seiner Gesamtheit demnach nicht zur Realisierung des Datentransfer-Patterns geeignet. Einige der Ideen und Komponeten daraus lassen sich jedoch für eine eigentsändige Implementierung verwenden. Sie werden in Kapitel 6 wieder aufgegriffen.

5.3 Champagne als Dienst

In [MJHM09] werden u.a. Möglichkeiten zum Einsatz von Champagne als Service-Aufruf diskutiert. Falls die dazu notwendigen Erweiterungen umgesetzt würden, wäre eine Verwendung von Champagne als Dienst beispielsweise für die Realisierung des Datentransfer-Patterns denkbar. Der Aufruf des Datentransfer-Patterns könnte dann mit einem Aufruf des Champagne-Dienstes verknüpft werden, der mit Hilfe einer speziellen Änderungsbeschreibung den Datentransfer ausführt. Dazu müssten etwaige Quelladapter so modifiziert werden, dass sie Datenänderungen nicht permanent überwachen. Stattdessen soll bei Aufruf des Dienstes das angegebene Quellsystem so behandelt werden, als wären alle zu übertragenden Daten neu erzeugt bzw. verändert worden. Der Quelladapter müsste also Änderungsbeschreibungen für jeden einzelnen Datensatz zu jedem Quellsystem generieren, der zu einem Zielsystem übertragen werden soll. Mit Hilfe entsprechender Propagationsskripte könnte daraufhin der Datentransfer realisiert werden. Diese Anpassung wäre sehr umständlich und vermutlich auch sehr ineffizient in ihrem Laufzeitverhalten, da der Datentransfer über den Umweg von einzelnen Änderungsbeschreibungen realisiert würde. Außerdem ließen sich die Propagationsprozesse, die den Datentransfer durchführen, nicht ohne weiteres durch den Workflow überwachen.

5.4 Champagne als Ergänzung eines Workflows

Es besteht die Möglichkeit, Champagne als Gesamtsystem parallel zu einer Workflowausführung zu betreiben. Eine Anwendung fände dies z.B. bei der in [Dor11] beschriebenen Kopplung von PANDAS mit MATLAB[1]. Hierbei werden an bestimmten Stellen abwechselnd PANDAS- und MATLAB-Simulationen durchgeführt. Die erzeugten Daten müssen dabei in

[1] http://www.mathworks.de/products/matlab/

jedem Simulationsschritt zwischen dem PANDAS-Format (Datenbank) und dem MATLAB-Format (CSV-Datei) hin- und herkonvertiert und übertragen werden. Für den Kontext dieser Datenübertragungen könnte Champagne eingesetzt werden. Dadurch könnten, die richtigen Propagationsskripte vorausgesetzt, fertig berechnete Ergebnisse sofort an das DM-System der anderen Simulation propagiert werden. Dieser Ansätz bedarf jedoch weiterer Evaluierungen, die hier nicht vorgenommen werden, beispielsweise stellt sich die Frage nach der Synchronisation der Simulationen mit den Propagationsprozessen.

5.5 SIMPL zur Datenänderungspropagation

Bisher wurden die Möglichkeiten und Einschränkungen beleuchtet, wie sich Champagne zur Erweiterung von SIMPL und insbesondere zur Realisierung des Datentransfer-Patterns verwenden lässt. In diesem Abschnitt wird diskutiert, ob sich SIMPL bzw. einzelne Komponenten oder Konzepte daraus zur Umsetzung von Datenänderungspropagation eignet.

Da SIMPL eine Erweiterung eines WfMS ist, muss die Frage zunächst lauten, inwiefern sich Champagne mit Workflowtechnologie sinnvoll kombinieren lässt. Wird ein WfMS, insbesondere eine Workflow Execution Engine und eine entsprechende Modellierkomponente, an Champagne angebunden, können Workflows innerhalb des Systems ausgeführt werden. Propagationsskripte könnten als Workflows modelliert und anschließend als Dienste deployed werden. Mit Hilfe der DM-Aktivitäten vom SIMPL könnten die Propagationsskripte dann ohne Umweg über Warteschlange und Adapter direkt auf Datenquellen zugreifen und somit die Effizienz der Propagationsprozesse möglicherweise steigern, sofern das Konzept der Änderungsbeschreibungen mit den DM-Aktivitäten vereinbart werden kann. Allerdings wäre zu untersuchen, ob die Synchronisation der Propagationsprozesse innerhalb eines Workfows zu bewerkstelligen ist.

Durch übergeordnete Workflows könnte man die gesamte Funktionalität des Propagationsmanagers modellieren und mit Hilfe des WfMS umsetzen. Dadurch stünden für den Propagationsmanager alle Vorteile einer Workflowausführung, insbesondere die Überwachungskomponenten *Auditing*, *Monitoring* und *Provenance* zur Verfügung.

Betrachtet man die SIMPL-Konzepte im Einzelnen, so ließe sich das Prinzip der DM-Patterns auch auf den Abhängigkeitsmanager von Champagne anwenden. Anstatt alle Propagationsskripte manuell zu erstellen, könnte man bestimmte, häufig benötigte Teile als Muster oder Vorlagen auslagern, um sie anschließend in anderen Skripten wiederzuverwenden.

Außer den in Abschnitt 5.1 vorgestellten Gemeinsamkeiten der Komponenten beider Systeme sind zunächst keine weiteren, sinnvollen Erweiterungen von Champagne durch Konzepte aus SIMPL ersichtlich.

6 Implementierung

In diesem Kapitel werden mögliche Implementierungen einer Komponente untersucht, die Datentransfers zwischen verschiedenen Datenquellen eines Simulationsworkflows automatisiert. In Abschnitt 3.3 wurde das Konzept der DM-Patterns als Teil des SIMPL-Rahmenwerks vorgestellt. Der automatisierte Datentransfer soll nun im Rahmen eines Datentransfer-Patterns implementiert werden, das die in Abschnitt 2.5.2 beschriebenen, vorhandenen Dienste und Workflow-Fragmente (im Folgenden *Akteure* genannt) verwendet. Diese können ihrerseits zur Durchführung des Datenransfers die Funktionen von SIMPL verwenden. Statt Workflow-Fragmente werden in diesem Kapitel fertige Workflows betrachtet, die wie ein Dienst aufgerufen werden.

Die Realisierung des Datentransfer-Patterns soll in Form eines *Transferpattern-Managers* implementiert werden. Zunächst werden die Anforderungen an ein Datentransfer-Pattern für den Einsatz in Simulationsworkflows formuliert. Daraus werden anschließend benötigte Architekturbausteine abgeleitet. Schließlich wird die erarbeitete Architektur des Transferpattern-Managers konkretisiert und ihre prototypische Umsetzung beschrieben.

6.1 Anforderungen

Bevor verschiedene Möglichkeiten zur Implementierung eines Transferpattern-Managers untersucht werden, wird zunächst untersucht, was das Datentransfer-Pattern im einzelnen leisten und wie es eingesetzt werden soll.

Ein Workflow-Entwickler soll in der Lage sein, ein Datentransfer-Pattern in einem Simulationsworkflow einzusetzen, um damit einen Datentransfer zu modellieren. Es wäre wünschenswert, diesen Transfer direkt zwischen den Datenquellen stattfinden zu lassen, dies wird jedoch von der Implementierung der Akteure vorgegeben. Die zwischen zwei Simulationsschritten, z.B. Berechnungen durch zwei verschiedene Programme, notwendigen Datentransfers sollen dabei durch einen Zwischenschritt modelliert werden können, der durch das Pattern repräsentiert wird. Dabei sollen vom Entwickler nur die Quell- und Zieldatencontainer des Transfers als Eingabeparameter angegeben werden müssen. Wie bereits in Abschnitt 3.2 beschrieben, kann ein Datencontainer anhand einer *data container reference variable* eindeutig identifiziert und einer Datenquelle zugeordnet werden.

Zur Laufzeit sollen beim Aufruf des Patterns die richtigen Akteure für den Transfer anhand der Eingabeparameter ausgesucht und aufgerufen werden. Eventuell auftretende Rückmeldungen der aufgerufenen Akteure sollten registriert und an den Workflow geleitet werden.

6.2 Metadaten

Das Datentransfer-Pattern muss anhand der ihm übergebenen Eingabeparameter, die Quell- und Zielcontainer enthalten, den richtigen Akteur für die Operation aussuchen und aufrufen. Um den richtigen Akteur für eine Quellen-Ziele-Kombination finden zu können, muss eine Zuordnung anhand von Metainformationen erfolgen. Eine Filterung aller Anfragen zu Beginn des Prozesses, wie es momentan bei Champagne vorgesehen ist, ist für das Datentransfer-Pattern nicht wünschenswert, da der Aufruf aller bekannten Dienste die Effizienz der Workflowausführung dramatisch beeinflussen würde und die vorhandenen Dienste auch nicht zwingend eine Überwachung der Eingabeparameter zur Filterung von Aufrufen implementieren.

Da DM-Patterns als Komponente eines SIMPL-sWfMS konzipiert sind, kann die bereits vorhandene Resource Management Komponente von SIMPL zur Speicherung der Zuordnungsinformationen verwendet werden. Eine einzelne, zusätzliche Tabelle (*Transfer Registry*) in der Datenbank genügt, um alle vom Transferpattern-Manager benötigten Metainformationen zu verwalten.

Alle einsetzbaren Akteure realisieren zusammen eine endliche Menge an Transferoperationen, wobei jeder Akteur genau eine Transferoperation auf den stets selben DM-Systemen durchführt. Es wird zusätzlich davon ausgegangen, dass für jede Quellen-Ziele-Kombination maximal ein Akteur existieren darf. Dies stellt eine eins-zu-eins Beziehung zwischen Akteuren und Systemkombinationen dar. Die Zuordnung kann daher mittels eines einzigen Datensatzes beschrieben werden, wobei sowohl der Akteur, als auch die Systemkombination als Primärschlüssel verwendet werden kann. Das Schema für diese Tabelle besteht also aus einem Attribut *Sources*, das die Quellen und Ziele eines Transfers beispielsweise im XML-Format enthält, sowie einem Attribut *Actor*, dass die Endpunktreferenz und WSDL-Operation eines im Service Bus bekannten Akteurs in Form einer Zeichenkette speichert.

6.3 Entwurfs- und Laufzeitkomponenten

Um das neue DM-Pattern prototypisch bei der Modellierung eines Simulationsworkflows verwenden zu können, wird, ähnlich wie bereits bei SIMPL, die sWF-Modeler Komponente des sWfMS um eine neue Aktivität erweitert, so dass der Entwickler das Pattern in den Workflow platzieren kann. Die neue Aktivität bekommt die Bezeichnung *AutoTransfer*. Die Quellen-Ziele-Kombination werden in Form von Eingabeparameter, die im gleichen Schema angegeben werden, wie das in der Transfer Regitry verwendeten (z.B. im XML-Format), an die Aktivität übergeben. Als weitere Überlegung könnte man eine zusätzliche GUI-Komponente zur Verfügung stellen, mit deren Hilfe in den Metadaten vorhandene, gültige Quellen-Ziele-Kombinationen angezeigt werden können.

Die Ausführungsengine muss analog zum DM Activity Execution Plug-In erweitert werden, damit die AutoTransfer-Aktivität vom sWfMS ausgeführt werden kann. Was noch fehlt, ist eine Kernkomponente (*Transfer Manager*) innerhalb des Service Bus, die die Zuordnung

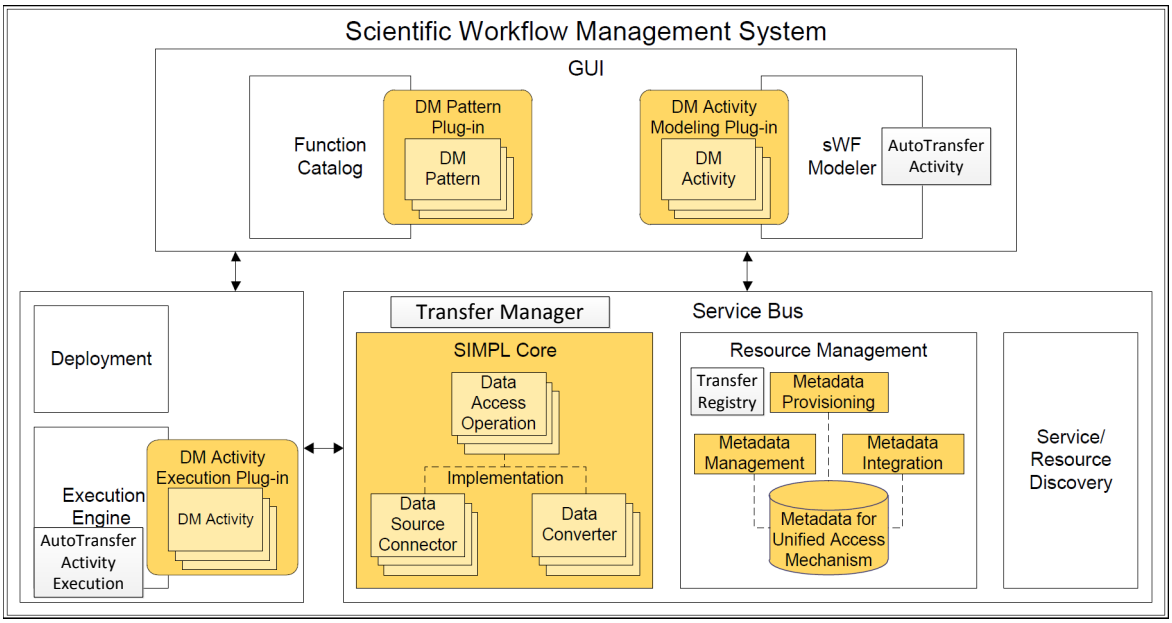

Abbildung 6.1: Architektur des Transferpattern-Managers (Vgl. Abbildung 3.1)

der Akteure anhand der Metainformationen in der Transfer Registry und deren Aufrufe übernimmt. Eine ähnliche Funktionalität wird auch durch den Propagationsmanager von Champagne, der einer Änderungsbeschreibung die richtigen Propagationsskripte zuordnet, zur Verfügung gestellt. Der Transfer Manager wird ähnlich wie der SIMPL-Core im Service Bus des sWfMS deployed und von der Ausführungsengine aufgerufen. Abbildung 6.1 zeigt analog zu Abbildung 3.1, wie ein SIMPL-sWfMS um den Transferpattern-Manager erweitert wird. Die Interaktion des Transfer Managers mit der Transfer Registry und der Ausführungsengine wird im nächsten Abschnitt genauer beschrieben.

6.4 Funktionsweise des Transferpattern-Managers

Beim Aufruf einer AutoTransfer-Aktivität wird von der Ausführungsengine zunächst der Transfer Manager aufgerufen und die Eingabeparameter an ihn übergeben. Dieser versucht anhand der Metainformationen der Transfer Registry einen passenden Akteur zu finden, der anschließend aufgerufen wird. Dabei ist zu beachten, dass die Struktur der Eingabeparameter für die Quell- und Zielcontainer keine Reihenfolge vorgibt, sie müssen lediglich voneinander unterscheidbar sein. Der Transfer Manager muss daher eine Analyse des Einbageparameters vornehmen, bei der Quell- und Zielcontainer identifiziert und ggf. für den direkten Vergleich mit der Transfer Registry sortiert werden.

Findet der Transfer Manager einen zu dem Eingabeparameter passenden Akteur, so muss dieser im nächsten Schritt aufgerufen werden. Dies geschieht im, wie zu Anfang dieses Kapi-

tels erwähnt, als Dienstaufruf. Betrachtet man zusätzlich den Fall von Workflow-Fragmenten als Akteure, so müssen diese mittels Prozessfragmentierungstechnologie in den laufenden Simulationsworkflow integriert werden. Dieser Fall wird hier jedoch nicht behandelt.

Als letztes Glied in der Funktionskette sollen etwaige, von einem Akteur generierte Nachrichten vom Transfer Manager empfangen und an den Simulationsworkflow weitergeleitet werden können. Für die Zukunft soll eine selbstständige, akteurspezifische Handhabung von Nachrichten ohne Weiterleitung an den Workflow in Form einer Erweiterung der Datentransfer-Komponente realisierbar sein.

7 Ergebnis und zukünftige Arbeiten

In den vergangenen Kapiteln wurde das SIMPL-Rahmenwerk sowie das Champagne-System vorgestellt und es wurde untersucht, in wie fern sich die beiden Konzepte sinvoll miteinander kombinieren lassen. Im Vordergrund stand dabei die Realisierung eines Datentransfer-Patterns, welches in einen Simulationsworkflow eingebettet wird und Daten von Quellsystemen auf Zielsysteme automatisch überträgt. Dabei hat sich hereausgestellt, dass Champagne in seiner jetzigen Form nicht dazu geeignet ist, in einen Workflow integriert zu werden. Außerdem ist der Großteil der Champagne-Funktionalität, wie die Synchronisation der Propagationsprozesse, für die Aufgabenstellung nicht relevant.

Stattdessen wurde der Einfachheit halber eine eigentständige Implementierung in Form eines Transferpattern-Managers gewählt, der ein SIMPL-sWfMS um die dafür benötigen Komponenten erweitert. Obwohl dabei keine Komponenten aus Champagne direkt übernommen wurden, so halfen die Ideen und Konzepte hinter der Datenänderungspropagation mit Champagne dabei, ein Architekturmodell für den Transferpattern-Manager zu entwickeln.

Für weitere Arbeiten bieten sich folgende, in dieser Arbeit nicht ausführlich betrachteten Themen an:

- Das Architekturmodell von SIMPL besitzt keine Komponente zur Bearbeitung der in der Resource Management-Komponente verwalteten Daten. Dazu wird momentan ein eiinfaches Webinterface verwendet. Hierzu könnte weiterhin untersucht werden, ob sich der Abhängigkeitsmanager von Champagne evtl. für diese Aufgabe eignet.
- Die Kopplung von PANDAS mit MATLAB (vgl. Abschnitt 5.4) unter Verwendung von Champagne zum Austausch der Simulationsdaten könnte näher untersucht werden.
- Bei der Implementierung des Transferpattern-Managers wurden Akteure nur als Dienstaufrufe betrachtet. Die als Workflow-Fragment vorliegenden Transferoperationen könnten jedoch auch in einen Workflow eingebettet ausgeführt werden. Dadurch ließen sich die Vorteile einer Workflowausführung beim Datentransfer nutzen.
- Neben des in dieser Arbeit betrachteten Datentransfer-Patterns gibt es noch eine Reihe anderer Patterns, die nocht nicht realisiert worden sind, z.B. das *Join-*, *Merge-* oder das *Split*-Pattern. Die Realisierung von DM-Patterns ist ein wesentlicher Bestandteil der Funktionalität von SIMPL und sollte daher besondere Aufmerksamkeit genießen.

Literaturverzeichnis

[AK03] W. M. P. van der Aalst, A. Kumar. XML-Based Schema Definition for Support of Interorganizational Workflow. In *Information Systems Research*, pp. 23–46. 2003. (Zitiert auf Seite 26)

[AMA06] A. Akram, D. Meredith, R. Allan. Evaluation of BPEL to Scientific Workflows. *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 2006. (Zitiert auf Seite 12)

[AVK01] W. M. P. van der Aalst, H. M. W. Verbeek, A. Kumar. Verification of XRL: An XML-based Workflow Language. In *Proceedings of the 6th International Conference on CSCW in Design*, pp. 427–432. NRC Research Press, 2001. (Zitiert auf Seite 26)

[bpe] Web Services Business Process Execution Language Version 2.0 - OASIS Standard. URL http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf. (Zitiert auf den Seiten 7 und 12)

[CHRM01] C. Constantinescu, U. Heinkel, R. Rantzau, B. Mitschang. SIES - An Approach for a Federated Information System in Manufacturing. In *Proceedings of the International Symposiumon Information Systems and Engineering (ISE)*, pp. 269–275. CSREA Press, 2001. (Zitiert auf Seite 26)

[dom] Document Object Model (DOM). URL http://www.w3.org/DOM/. (Zitiert auf Seite 25)

[Dor11] R. Dormien. *Service-Bus-Erweiterung um Pandas-basierte Simulationen in Workflows zu nutzen*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf Seite 34)

[EUL09] H. Eberle, T. Unger, F. Leymann. *Process Fragments. In: On the Move to Meaningful INternet Systems: OTM 2009, Part I*. Springer, 2009. (Zitiert auf Seite 21)

[G+11] K. Görlach, et al. *Guide to e-Science*. Springer, 2011. (Zitiert auf den Seiten 5, 10 und 11)

[Hei11] U. Heinkel. Änderungspropagation für autonome und heterogene Informationssysteme, 2011. Dissertation, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart. (Zitiert auf den Seiten 7 und 23)

[KKL+05] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, I. Trickovic. WS-BPEL Extension for People BPEL4People, 2005. (Zitiert auf Seite 12)

[LR00] F. Leymann, D. Roller. *Production Workflow - Concepts and Techniques*. Prentice Hall PTR, 2000. (Zitiert auf den Seiten 7 und 10)

[MJHM09] J. Minguez, M. Jakob, U. Heinkel, B. Mitschang. A SOA-based Approach for the Integration of a Data Propagation System, 2009. (Zitiert auf Seite 34)

[MRZ11] J. Minguez, P. Reimann, S. Zor. Event-driven Business Process Management in Engineer-to-Order Supply Chains. In *Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design*, pp. 1–8. IEEE, 2011. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2011-42&engl=0. (Zitiert auf Seite 7)

[RCHM02] R. Rantzau, C. Constantinescu, U. Heinkel, H. Meinecke. Champagne: Data Change Propagation for Heterogeneous Information Systems. *18th VLDB Conference*, 2002. (Zitiert auf den Seiten 5 und 27)

[RRS+11] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL - A Frameworrk for Accessing External Data in Simulation Workflows, 2011. (Zitiert auf den Seiten 5, 7, 10, 13, 14, 17, 18 und 21)

[soa] SOAP Version 1.2 - W3C Recommendation (Second Edition) 27 April 2007. URL http://www.w3.org/TR/soap/. (Zitiert auf Seite 10)

[TDG07] I. Taylor, E. Deelman, D. Gannon. *Workflows for e-Science - Scientific Workflows for Grids*. Springer, 2007. (Zitiert auf den Seiten 7 und 10)

[VHA02] H. M. W. Verbeek, A. Hirnschall, W. M. P. van der Aalst. XRL/Flower: Supporting Interorganizational Workflows using XRL/Petri-net Technology. In *Lecture Notes in Computer Science: Web Services, E-Business, and the Semantic Web, CAiSE International Workshop (WES 2002)*, pp. 93–108. Springer Verlag, Berlin, 2002. (Zitiert auf Seite 26)

[xml] Extensible Markup Language (XML) 1.0 (Fifth Edition). URL http://www.w3.org/TR/REC-xml/. (Zitiert auf Seite 9)

[xpa] XML Path Language (XPath) Version 1.0 - W3C Recommendation 16 November 1999. URL http://www.w3.org/TR/xpath/. (Zitiert auf Seite 9)

[xpd] XPDL Support and Resources. URL http://www.wfmc.org/xpdl.html. (Zitiert auf Seite 10)

[xqu] XQuery 1.0: An XML Query Language (Second Edition) - W3C Recommendation 14 December 2010. URL http://www.w3.org/TR/xquery/. (Zitiert auf Seite 9)

[xsl] XSL Transformations (XSLT) Version 2.0 - W3C Recommendation 23 January 2007. URL http://www.w3.org/TR/xslt20/. (Zitiert auf Seite 9)

Angegebene Links wurden zuletzt überprüft am: 28.09.2011

Erklärung

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

(Christian Ageu)