Institut für Parallele und Verteilte Systeme
Abteilung Anwendungssoftware
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3235

Abstraktionsunterstützung für die Definition des Datenmanagements in Simulationsworkflows

Stavros Aristidou

Studiengang:	Informatik
Prüfer:	PD. Dr. rer. nat. Holger Schwarz
Betreuer:	Dipl.-Inf. Peter Reimann
begonnen am:	18. August 20011
beendet am:	17. Februar 2012
CR-Klassifikation:	D.2.11, H.2.5, H.4.1, I.6.7

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Seit langem wurden Workflows als ein Mittel zur Unterstützung der Geschäftsprozesse in Unternehmen verwendet [LR00]. Seit kurzem werden Worklows auch im Bereich der wissenschaftlichen Simulationen eingesetzt.[TDG07] Simulationsanwendungen beinhalten viele komplizierte Berechnungen und Datenverwaltungsaufgaben. Der Zugriff, die Bereitstellung und die Generierung großer Datenmengen in heterogenen und verteilten Umgebungen stellen eine große Herausforderung für die Zukunft dar [Gil07] [EC08]. Aufgrund dieser Problematiken wäre die Entwicklung einer konsolidierten und integrierten Datenmanagementabstraktion für die Wissenschaftler besonders sinnvoll und hilfreich [RRS+ny]. Das SIMPL-Rahmenwerk bietet eine solche generische und erweiterbare Datenmanagment-und Datenbereitstellungsabstraktion für Simulationsworkflows an. Dieses Rahmenwerk ermöglicht es, über eine Erweiterung der Workflow Sprache WS-BPEL in Datenmanagementaktivitäten, direkt von einem Workflow aus, auf externen Datenquellen direkt und nahtlos zuzugreifen. Diese Arbeit befasst sich mit dem Aspekt der Definition des Datenmanagements in Simulationsworkflows. Ziel ist es die Definition des Datenmanagements für bestimmte Simulationsanwendungen zu vereinfachen.

1.1 Motivation und Aufgabe dieser Arbeit

Im Rahmen dieser Arbeit werden zusätzliche Datenmenagementpatterns, die den Wissenschaftler als Templates für ausführbare Workflow-Fragmente zur Verfügung stehen sollen entwickelt. Dadurch wird die Definition des Datenmanagements für weitere Simulationsanwendungen vereinfacht. Aus konkreten Anwendungsszenarien sollen zunächst geeignete Datenmanagementpatterns herausgearbeitet und identifiziert werden. Die identifizierten Datenmangementpattens sollen definiert und anschließend auf ausführbare Workflow-Fragmente, etwa auf Datenmanagmentaktivitäten oder Service Aufrufe, transformiert werden. Für diese Transformationen, soll ein Rahmenwerk entstehen, welches auf Abbildungsregeln und Metadaten über die während der Ausführung eines Simulationsworkflows involvierten Datenquellen basiert.

1.2 Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2-Grundlagen: In diesem Kapitel werden grundlegenden Begriffe, die für

das Verständnis dieser Arbeit benötigt werden, erläutert. Darüber hinaus werden verschiedenen Simulationsworkflowmanagementsysteme und Simulationsrahmenwerke kurz vorgestellt.

Kapitel 3-Das SIMPL Rahmenwerk: Stellt das Rahmenwerk SIMPL vor und erklärt dessen Architektur und Funktionsweise.

Kapitel 4-Simulationsworkflows: Im Rahmen dieses Kapitels werden die verwendeten Anwendungsszenarien, aus denen die Datenmanagementpatterns identifiziert werden, vorgestellt.

Kapitel 5-Datenmanagementpatterns: Die identifizierte Datenmanagementpattern werden in diesem Kapitel formalisiert und erläutert.

Kapitel 6-Abbildung der Datenmanagementpatterns auf ausführbare Workflow-Fragmente: Stellt den Ansatz für die Transformation der Datenmanagementpatterns auf ausführbaren Workflow-Fragmente vor.

Kapitel 7-Zusammenfassung und Ausblick: Dieses Kapitel fasst diese Arbeit zusammen.

2 Grundlagen

In diesem Kapitel werden einige Begriffe erläutert, die zum Verständnis dieser Arbeit benötigt werden. Als erstes wird über das Konzept der Dienste diskutiert und der Begriff der Webservices als eine bekannte Realisierung dieses Konzepts erläutert. Danach werden wir einen Blick auf die Welt der Workflowtechnologie werfen und die Workflow-Sprache WS-BPEL kurz beschreiben. Weiterhin werden wir über die verschiedene Workflow-Arten diskutieren und einige Simulationsworkflowmanagementsysteme und -Rahmenwerke kurz vorstellen.

2.1 Service Oriented Architecture

Das Kapitel 2.1 baisert auf [SCF+05]. Andere Quellen werden explizit angegeben. Die **S**ervice **O**riented **A**rchitecture (SOA) ist ein Softwarearchitekturmuster, die aus kleinen, selbständigen und ausführbaren Programme besteht [SCF+05]. Die Zusammensetzung dieser Programme, die Services oder Dienste heißen, soll eine bestimmte Funktionalität realisieren. Hauptziel ist es, existierende Software miteinander kommunizieren zu lassen, um die vom Benutzer gewünschten Funktionalität zur Verfügung zu stellen. Heutzutage ist die Anwendung der SOA Technologie, sowohl in dem wirtschaftlichen als auch in dem wissenschaftlichen Bereich sehr bedeutend [TDG07]. Einfache Beispiele sind die Buchung einer Reise oder die Durchführung eines bestimmten wissenschaftlichen Experiments. Im Folgenden werden die wesentliche Merkmale der SOA Technologie aufgelistet:

- Lose Kopplung der Dienste
- Service\Dienstleistungs-Vertrag
- Abstraktion
- Wiederverwenderbarkeit der Dienste
- Zustandslosigkeit der Dienste

Im Rahmen der SOA gibt es drei Akteure: der *Dienstanbieter*, *der Dienstnutzer* und das *Service-Verzeichnis*. Der Dienstanbieter bietet die Dienste an, indem er diese durch eine abstrakte Art und Weise beschreibt und im Service-Verzeichnis veröffentlicht. Der Servicenutzer kann dort nach einem angebotenen Dienst, die eine bestimmte Funktionalität realisiert suchen und bekommt die für ihn passende Dienstbeschreibungen vom Service-Verzeichnis zurück. Der Dienstnutzer kann die von dem Service-Verzeichnis gelieferten Servicebeschreibung nutzen. Das Zusammenspiel der drei Akteuren ist auf Abbildung 2.1 dargestellt.

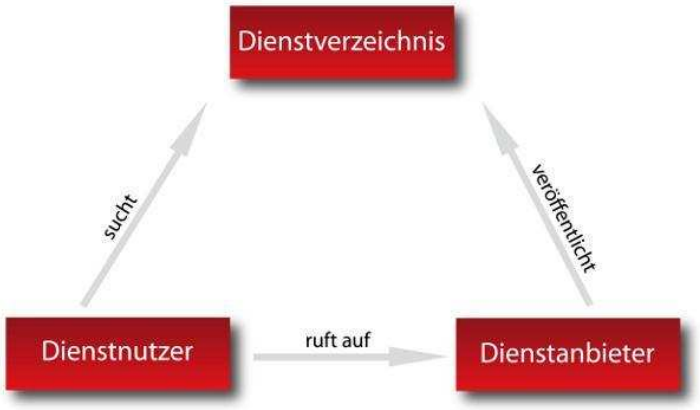


Abbildung 2.1: Das SOAP Dreieck [1]

2.1.1 Webservices

Zur Orchestrierung von Diensten werden Webservices eingesetzt. Unter einem *Webservice* versteht man *ëine Softwareanwendung, die mit einem URI eindeutig identifizierbar ist und eine Schnittstelle hat, die als XML Artefakt definiert, beschrieben und gefunden werden kann. Diese Softwareanwendung unterstützt die Interaktion mit anderen Webservices innerhalb eines Netzwerkes unter der Verwendung XML-basierten Nachrichten durch den Austausch über Internet-basierte Protokolle"* [2]. Einer Webservice definiert eine Menge von Operationen, die vom Benutzer aufrufbar sind. Die Webservices können entweder synchron oder asynchron aufgerufen werden. Bei einem synchronen Aufruf sendet der Klient seine Anfrage bzw. seine Nachricht an den Empfänger und bekommt sofort eine Antwort bzw. eine Nachricht zurück. Bei einem asynchronen Aufruf schickt der Klient seine Anfrage und wartet auf eine Antwort des Empfängers. Zur Realisierung dieser Klient-Server basierte Kommunikation, sind zwei Standards unverzichtbar. Das *Simple Object Access Protokoll* und die *XML basierte Sprache WSDL*.

Zur Standardisierung des Austausches von Nachrichten wurde das Simple Objekt Access Protokoll (SOAP) eingeführt [W3C07]. Über dieses XML-basiertes Protokoll werden Nachrichten zwischen Webservices innerhalb eines Netzwerkes ausgetauscht. Eine SOAP-Nachricht besteht aus einem Umschlag, der den Kopf und den Körper der Nachricht enthält [Wag11]. Der Kopf kann zusätzliche Transportinformationen beinhalten wie z.B Authorisierungs- und Verschlüsselungsinformationen. Im Körper werden die aufzurufende Methode des Empfängers sowie deren Eingabeparameter spezifiziert. Nachdem die Methode des Empfängers aufgerufen wurde wird das Ergebnis über eine SOAP-Nachricht an dem Sender zurückgeschickt, wobei diesmal im Körper der SOAP-Nachrichten die Ergebnisse gespeichert sind. Welche aufzurufende Methoden dem Sender zur Verfügung stehen, über

[1] http://bgoll.de/wp-content/uploads/2010/05/soa_dreieck.png
[2] http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211

welche tatsächliche Internet Adresse und über welches Transportprotokoll der Empfänger zu erreichen ist wird durch die Web Service Definition Language (WSDL) definiert.

2.1.2 Web Service Definition Language

In [SCF+05] wird die *Web Service Definition Language (WSDL)* als ein durchgesetztes und akzeptiertes Standard für eine SOA verstanden. WSDL ist eine XML basierte Sprache, die die Funktionalität eines Webservice spezifiziert, die Zugriffsmöglichkeiten auf einen Webservice beschreibt und die Informationen über den Ort, auf den sich die Web Service befindet, enthält. Eine WSDL-Datei besteht aus sieben Elementen:

Schema: Aufgrund der Tatsache, dass WSDL eine XML basierte Sprache ist werden zum Austausch von Nachrichten XML-Elemente verwendet. Dieses Schema kann innerhalb eines <types> Elements definiert werden. Es besteht auch die Möglichkeit, das Schema aus externen XSD-Dateien in die WSDL Datei zu importieren.

Message: Der Inhalt einer Nachricht wird über <message> Elemente definiert. Hierbei wird auf das zuvor spezifizierte Schema referenziert.

Port Typ: In einem <port type> werden die in einer Message definierten Operationen des Webservice, die aus einer Eingangs- und Ausgangsnachricht bestehen, beschrieben.

Binding: Über das <binding> wird das Transfer-Protokoll, auf dessen Basis die Nachrichten ausgetauscht werden beschrieben.

Port: Ein <Port> spezifiziert die tatsächliche URL (Endpoint), der ausgetauschten Nachrichten.

Service: Das <service> Element bündelt alle Ports. Damit können Operationen eines Services auf mehreren Endpunkten verstreut werden.

Die Orcherstrierung und Ausführung der Webservices wird von der Workflow-Technologie unterstützt. Diese Thematik wird der Gegenstand des nächsten Kapitels sein.

2.2 Workflowtechnologie

In den letzten Jahren haben sich Workflow Produkte in der Industrie als wichtiger Bestandteil der IT-Infrastruktur etabliert [LR00]. Durch die Verwendung von Workflows möchte man bestimmte Arbeitsabläufe automatisieren. Ein Arbeitsablauf besteht aus einzelnen Schritten. Diese einzelne Schritte werden als Aktivitäten bezeichnet. Ein Beispiel eines solchen Arbeitsablaufs ist die Auftragsabwicklung in einem Unternehmen. Die rechnergestützte Orchestrierung und Ausführung solcher Arbeitsabläufe wird in drei Dimensionen gespaltet:

Wie: In dieser Dimension wird festgelegt, welche Aktivitäten und mit welcher logischen Reihenfolge ausgeführt werden sollen.

Wer: In dieser Dimension wird bestimmt, welche Person oder welches Programm aus der Organisation eines Unternehmens in der Lage ist, eine konkrete Aktivität auszuführen. Zur Ermittlung des geeigneten Benutzers oder Programmes kann für jede Aktivität eine Anfrage angegeben werden.

Womit: In dieser Dimension werden die Mittel aus der IT-Infrastruktur, die für die Ausführung der Aktivitäten verantwortlich sind, beschrieben.

Auf Abbildung 2.2 werden die drei Workflow-Dimensionen als Würfel dargestellt. In diesem dreidimensionalen Workflowraum wird die Ausführung eines Workflows als eine Folge von Punkten interpretiert. Der Treffpunkt der drei Dimensionen stellt die Ausführung einer einzelnen Aktivität von einer Person oder Programm über ein bestimmtes Mittel dar.

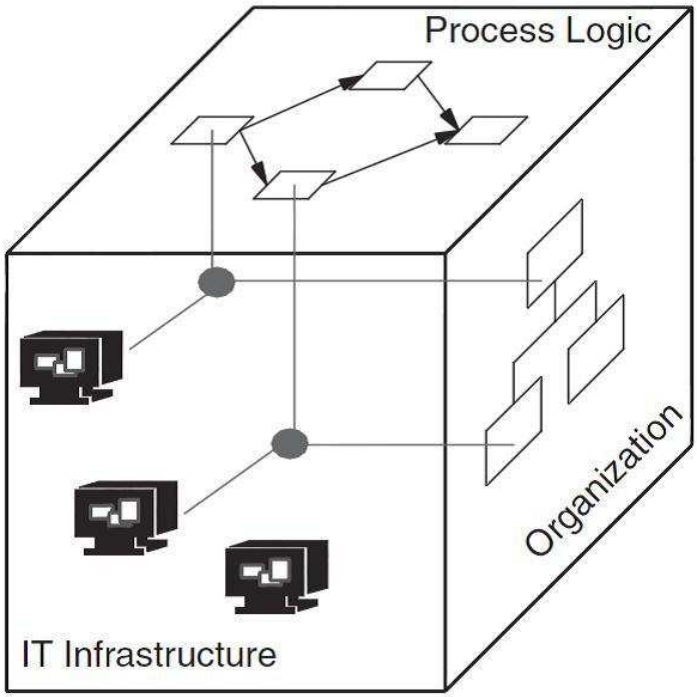


Abbildung 2.2: Die drei Workflow-Dimensionen [LR00]

Ein **Workflow Management System** (WfMS) beinhaltet Anwendungen, welche der Definition, der Verwaltung, der Ausführung und der Überwachung der Workflows dienen [Coa95]. Die Architektur eines WfMS's besteht hauptsächlich aus drei Komponenten [LR00]. Abbildung 2.3 zeigt die Architektur eines WfMS's.

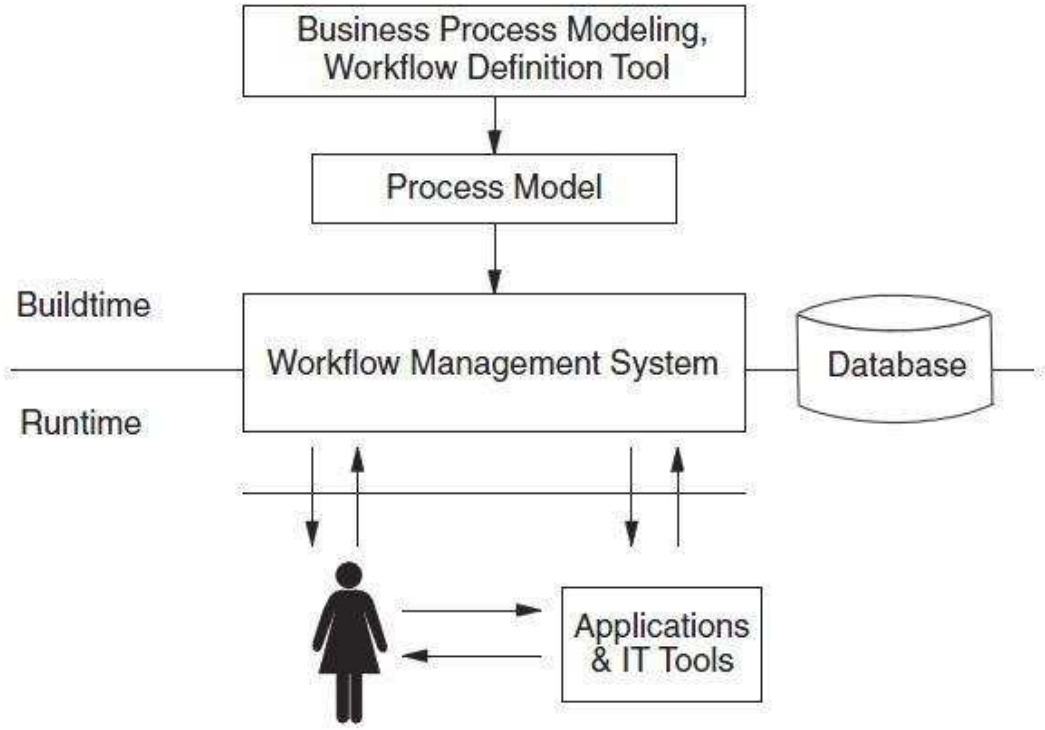


Abbildung 2.3: Die Architektur des WfMSs [LRoo]

Die Build-Time-Komponente fasst diejenige Komponenten zusammen, die zur Erstellung und Modellierung von Workflows verantwortlich sind. Weiterhin werden in der Build-Time die Ressourcen verwaltet.

Die Run-Time-Komponente führt die Workflowinstanzen zur Laufzeit aus.

Datenbank: In der Datenbank werden alle Daten der Build- und Run Time abgespeichert.

Metamodell: Diese Komponente wird in [LRoo] zusätzlich erwähnt. Hier wird die Struktur aller Ressourcen, die von einem WfMS unterstützt werden, definiert.

Es gibt verschiedenen Workflow-Sprachen, die die Struktur eines Workflows beschreiben. Man unterscheidet zwischen den Kontrollflussorientierten und den Datenflussorientierten Sprachen. Bei den Kontrollflussorientierten Sprachen liegt der Fokus auf die logische Reihenfolge der Ausführung der einzelnen Aktivitäten. Bei den Datenflussorientierten Sprachen steht die Datenversorgung der einzelnen Aktivitäten im Vordergrund. Im nächsten Teilkapitel werden wir uns mit der Worklflowsprache WS-BPEL, einer kontrollflussorientierten Sprache beschäftigen.

2.2.1 WS-BPEL

Die **Workflowsprache Webservices- Business Process Execution Language** (WS-BPEL) wurde von unterschiedlichen Unternehmen einwickelt und von OASIS standardisiert. Sie ist eine XML-basierte Sprache und gilt als Standard-Sprache zur Modellierung und Orchestrierung von Workflows [OAS]. WS-BPEL ist eine Kontrollflussorientierte Sprache. Zum Speichern von Daten verwendet sie Variablen und sie ist für die Orchestrierung von Webservice-Aufrufen zuständig (siehe Kapiel 2.2.1).

Einer BPEL-Prozess wird wie folgt aufgebaut:

- **Import:** Hier werden die WSDL-Datei und das XML-Schema im Prozess miteinbezogen.
- **Partner Links:** Tragen zur Einbindung der Webservices und Operationen bei.
- **Variablen:** Die globale Variablen und deren XML-Typen müssen definiert werden. In WS-BPEL wird zwischen drei verschiedenen XML-Typen unterschieden. Den WSDL-Nachrichten, den XML-Schema-Elementen und schließlich den XML-Schema-Typen.
- **Logik:** Hier wird der Kontrollfluss beschrieben, der auf globalen Variablen und Partner Links referenziert.

In BPEL werden die einzelne Sprachelemente als Aktivitäten bezeichnet. Im folgenden werden die wichtigste Aktivitäten kurz beschrieben.

ASSIGN: Über Assign erfolgt die Variablenzuweisung. Über mehreren COPY-Anweisungen können innerhalb einer Aktivität mehrmalige Variablenzuweisungen stattfinden. Die Zuweisung <from> sagt aus, wessen Variable der Inhalt kopiert wird. Die Zuweisung <to> sagt aus, wo dieser Inhalt kopiert werden soll.

IF: Realisiert die logische Reihenfolge der Ausführung der einzelnen Aktivitäten. Dies erfolgt über die Evaluierung eines Booleschen Ausdrucks und den entsprechenden Durchlauf des if-then-else Zweigs.

FOREACH: Realisiert eine Schleife. Hierbei wird der Start- und Endwert des Zählers über einen Ausdruck einer Query-Sprache ausgewertet. Bei jedem Durchlauf wird es um eins hochgezählt bis der Endwert erreicht wird. Wird dieser Endwert erreicht dann wird die Schleife abgebrochen.

WHILE: Arbeitet wie die klassische *While-Schleife*. Sie läuft so lange durch bis eine konkrete Abbruchs-Bedienung erfüllt wird.

REPEAT UNTIL: Ähnelt sich der *While Schleife*. Es wird lediglich nach dem Durchlauf des Schleifenrumpfes evaluiert. Folglich läuft der Schleifenrumpf mindestens ein mal durch.

ONALARM/Wait: Hierbei wird der Ablauf eines Workflows vorläufig unterbrochen.

Nach dem Eintritt eines gewissen Zeitspanes oder Ereignisses wird der Ablauf fortgesetzt.

INVOKE: Ist für den Aufruf einer Webservice (siehe Kapitel 2.1.1) zuständig. Dieser muss zuvor mit einem Partner Link eingebunden worden sein. Es werden die aufzurufende Operation und der Inhalt der Nachricht, die aus der zuvor definierten globalen Variable entnommen wird, festgelegt. Der Inhalt der Antwort-Nachricht und deren Ergebnis wird auf eine Variable kopiert .

Die Business-Workflows, *die wissenschaftliche Workflows* und die *ETL-Workflows* stellen die drei Workflow-Arten dar [Wag11]. Im nächsten Abschnitt werden wir kurz die Business-Workflows beschrieben. Auf die andere Workflow-Arten wird es in den nächsten Teilkapiteln näher eingegangen.

Business-Workflows werden zur Modellierung von bestimmten Arbeitsabläufen in Unternehmen eingesetzt [LR00]. Der Versand von Waren nach Zahlungseingang ist ein Beispiel eines solchen Workflows. Jede Workflow-Aktivität kann von verschiedenen Akteuren ausgeführt werden, z.B durch einen Mitarbeiter oder automatisch von einem Programm. Es wird danach über alle offene Rechnungen iteriert und die Firmenkonten nach einer Zahlung überprüft. Falls die Anforderung an den Kunden beglichen wurde, wird die Ware an den Kunden geliefert. Falls dies nicht der Fall ist dann wird überprüft, ob eine Mahnung veranlasst wird. In diesem Fall geht es um Workflows, bei denen Entscheidungen und Handlungen eine sehr große Rolle spielen. Daher müssen oft nur relevante Daten in der WF-Runtime gehalten werden. Aufgrund dieser Tatsache muss eine große Datenmenge nicht in der Runtime gehalten werden. Die Laufzeit solcher Workflows ist in der Regel kurz. Daher ist deren parallele Ausführung besonders sinnvoll.

2.2.2 Wissenschaftliche Workflows und Simulationen

Wissenschaftliche-Workflows werden innerhalb der Natur- und Ingenieurwissenschaften eingesetzt [TDG07]. Hierbei geht es um die rechnergestützte Durchführung von wissenschaftlichen Experimenten oder Simulationen bzw. um das Datenmanagement in solchen rechnergestützten Durchführungen [Wag11]. Es gibt oftmals keine standardisierte Software, die die Durchführung von wissenschaftlichen Simulationen unterstützt. Daher werden diversen Programme eingesetzt, die zur Berechnung und Analyse der Experiment- und Simulationsdaten zusammenarbeiten. Aufgrund dieser Tatsache müssen oftmals Daten auf ein anderes Format transformiert und/oder von einer Ressource auf eine andere übertragen werden, damit eine weitere Verarbeitung dieser Daten ermöglicht wird. Heutzutage ist es möglich solche Prozesse mit Workflows zu modellieren und zu orchestrieren. Bei den wissenschaftlichen Workflows liegt der Schwerpunkt eher auf die Datenverarbeitung als auf das Treffen von Entscheidungen. Die Ausführung solcher Workflows kann prinzipiell über zwei Wege erfolgen. Es gibt wissenschaftliche Workflows, die die Simulationsdaten an den beteiligten Programmen übergeben. Diese Programme verarbeiten und legen dieser Simulationsdaten in externen Datenressourcen ab. Es gibt aber auch solche, die Simulationsdaten direkt auf den Workflow-Kontext laden. Dadurch kommt es zu einem großen Datenaufkommen innerhalb

der WF-Runtime. Aufgrund der Tatsache, dass bei den wissenschaftlichen Workflows der Fokus auf den Datenaustausch zwischen Aktivitäten, Programmen und Datenressourcen liegt, sind Datenflussorientierte Workflow-Sprachen passender als die Kontrollfluss-orientierte Sprachen. Da aber WS-BPEL als Standard zur Modellierung und Orchestrierung von Workflows gilt und weil sich der Datenfluss in WS-BPEL einbetten lässt hat sich WS-BPEL für die Ausführung von wissenschaftlichen Workflows durchgesetzt [ADR06] [A.S06] [CTE09]. Die Ausführung von wissenschaftlichen Workflows findet seltener und mit einem umfangreicheren Datenaufkommen als die Ausführung von Business-Workflows statt. Daher ist eine parallele Ausführung solcher Workflows nur in speziellen Fällen sinnvoll.

Simulationsworkflows

Wie im vorigen Abschnitt erwähnt schließt die Ausführung von wissenschaftlichen Workflows Simulationen ein. Unter einer *Rechnergestützte-Simulation* versteht man eine beliebige Rechner-implementierte Methode zur Analyse der Eigenschaften mathematischer Modellen [P.H90]. Auf der Abbildung 2.4 ist eine Beispielsimulation dargestellt. Zur Vorbereitung einer Simulation wird zuerst darauf geachtet, das zu simulierende Objekt in der realen Welt darzustellen. Die Durchführung von Simulationen zielt darauf ab, das Verhalten des zu simulierenden Modells unter Eingabe einer Menge von Parametern, z.B Umwelt-Variablen, zu studieren und zu analysieren. Ausgehend von einem konkreten Startzustand des Modells wird die Simulation innerhalb eines bestimmten Zeitintervalls durchgeführt. Daraus resultieren sich die Zwischenzustände-und der Endzustand des zu simulierenden Objekts. Nachdem die Simulation des Objekts erfolgreich beendet wird, kann man den Zustand des ursprünglichen Modells unter der Eingabe derselben Eingabeparameter prognostizieren, z.B wenn das Auto bei einer Geschwindigkeit von 80 km/h gegen einen konkreten Objekt zusammenprallt. Die Verwendung von Simulationen dient dazu, die Durchführung einer bestimmten Anzahl von Experimenten zu minimieren oder sogar zu vermeiden.

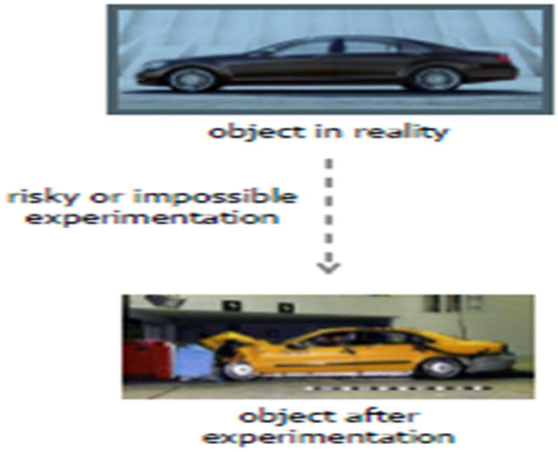


Abbildung 2.4: Beispiel einer Simulation anhand eines Autounfalls [Mü10]

In [S.H05] wird erwähnt, wie Simulationen eingesetzt werden können:

1. "Als eine Technik zur Ermittlung der Details eines Systems, das pragmatisch nicht als ein Experiment durchgeführt werden kann (das System ist zu komplex oder das Objekt ist nicht Verfügbar) Beispiel: Schwarzes Loch-Simulation in Astrophysik".
2. "Als ein heuristisches Werkzeug zur Entwicklung von Hypothesen, Modellen, und Theorien durch die Extraktion von Wissen über ein System, das über die Durchführung von mehreren Simulationsschritten gewonnen wird. Beispiel: Simulation der Dynamik innerhalb physikalischer Partikeln um neue Theorien über ihre Festigkeit aufzubauen".
3. "Als einer Ersatz für ein Experiment zur Durchführung von Experimenten, bei denen eine bestimmte Region von Parametern unzugänglich ist. Beispiel: Simulation einer Steuererhöhung von 100%."
4. "Als Werkzeug für die Durchführung von Experimenten: Simulationen können als Inspiration zur Durchführung eines Experiments dienen. Weiterhin können Simulationen zur Analyse von Experimenten beitragen."
5. "Als pädagogisches Werkzeug zum didaktischen Zwecken."

In [Mü10] werden Simulationsworkflows als eine Teilmenge der wissenschaftlichen Workflows angesehen. Das ist aber nicht streng zu interpretieren, denn das Beispiel zur Simulation eines Autounfalls hat auch einen wirtschaftlichen Hintergrund, obwohl es als einer Simulationsworkflow betrachtet wird. Nachdem man das geklärt hat, kann man den Begriff Simulationsworkflow definieren:

"Ein Simulationsworkflow ist ein wissenschaftlicher Workflow, der auf die Simulation eines Modells basiert und dessen Verhalten unter der Eingabe einer bestimmten Eingabemenge kalkuliert." [Mü10]

In [BIS+06] wird der Begriff Simulationsworkflow *im engeren Sinn wissenschaftlich orientiert angesehen*, weil seine Aktivitäten die Simulationsschritten eines Modells repräsentieren. Aber vor der Durchführung der eigentlichen Simulationsschritten müssen oft verschiedene Eingabedaten in der Simulation miteinbezogen werden. Aktivitäten, die zur Vorbereitung der Simulation dienen, werden oft als einer Teil des gesamten Prozesses modelliert. Weil diese Teile des gesamten Workflows implizite Simulationsschritten repräsentieren, werden als Simulationsworkflows in *weiteren Sinn betrachtet* .

Finite Element Methode

Viele Simulationsmodelle basieren im Engineering und in der Physik auf Differentialgleichungen. Im Folgenden werden wir wir den Begriff Finite Element Method (FEM) kurz erläutern. Die Beschreibung der FEM-Methode basiert auf [Dor11] und [ZTZ06].

Die *Finite Element Methode (FEM)* ist ein numerisches Verfahren, das zur Lösung

von partiellen Differentialgleichungen eingesetzt wird. Sie wird zur Lösung unterschiedlicher Problemstellungen in vielen wissenschaftlichen Bereichen verwendet. Für die verschiedene Problemstellungen können mehrere Ansätze, wie z.B der Ritz-oder der Galerkin Ansatz, verwendet werden. Bei dem Galerkin-Ansatz wird das Problem in einer endlichen Anzahl von Teilproblemen zerlegt. Diese endliche Teilproblemen, werden durch ein numerisches Verfahren vernetzt, um das gesamte Problem anzunähern. Diese kleinere endliche Teilprobleme heißen endliche Elemente und besitzen in der Regel eine einfache geometrische Form. Abbildung 2.5 zeigt ein Beispiel eines zweidimensionalen FEM-Modells. In diesem Beispiel hat man als geometrische Form der endlichen Elemente das Dreieck ausgewählt. Die Knoten stellen die einzelnen Elemente dar. Die einzelne Elemente werden nun über die Kanten unter der Anwendung eines numerischen Verfahrens miteinander verbunden, sodass der sogenannte Gitter gebildet wird. Zum Festlegen der Reihenfolge der Elementberechnung, werden die Elemente innerhalb dieses Gitters eindeutig beziffert.

Unter der Verwendung dieser endlichen Elemente und geeigneten numerischen Verfahren kann man das Modell annähern, indem man die einzelne Elemente in endlichen Zeitschritten t_0 bis t_n diskretisiert. Dazu werden an den Knotenpunkten Gleichungen aufgestellt, die die gesamte Lösung näherungsweise beschreiben. Der Zustand bzw.der Wert eines einzelnen Elements lässt sich zum Zeitpunkt t_i über eine Matrix von Gleichungen repräsentieren. Aus den Matrizen aller Elemente wird nun eine globale Matrixgleichung aufgestellt, die den Zustand bzw. den Wert des gesamten Modells zum Zeitpunkt t_i beschreibt. Der Zustand jedes einzelnes Elements zum Zeitpunkt t_i und damit der Zustand des gesamten Problems zum Zeitpunkt t_{i+1} werden anschließend über ein iteratives numerisches Verfahren berechnet. Dieses Verfahren wird wiederholt bis der Zeitschritt t_n erreicht wird . Die FEM-Methode wird im Bereich der Simulationen eingesetzt. In der Pandas-Simulation (siehe Kapitel 4.2) wird diese Methode verwendet, um die Umformung einer Knochenstruktur unter bestimmten Belastungen auf den Knochen zu simulieren. Die FEM-Methode findet auch im Bereich der Multi-Simulationen ihre Anwendung. Im Folgenden werden Beispiele von Multi-Simulationen aufgelistet:

- Multi-Domänen: Zur Durchführung der Simulation werden verschiedenen Fachbereiche, z.B Biomechanik und Biologie, miteinbezogen.
- Multi-Skalen: Die Simulation wird auf unterschiedlichen Zeit- und/oder Raumskalen durchgeführt.
- Multi-Physik: In der Simulation finden verschiedene physikalische Gesetze ihre Anwendung
- Multi-Tool: Für die Simulation werden verschiedene Werkzeuge eingesetzt.

Die Pandas-Matlab Kopplung (siehe Kapitel 4.3) stellt ein Beispiel einer Multi-Simulation dar. In diesem Fall werden verschiedenen Domäne einbezogen (Biomechanik und Biologie). Folglich wird die Simulation auf unterschiedliche Zeit- und Raumskalen durchgeführt. Darüber hinaus werden zwei unterschiedlichen Programme benutzt, also Pandas und Matlab.

[3]http://www.ite.uni-stuttgart.de/forschung/projekte/elfe/fem2d/fem2d.jpg

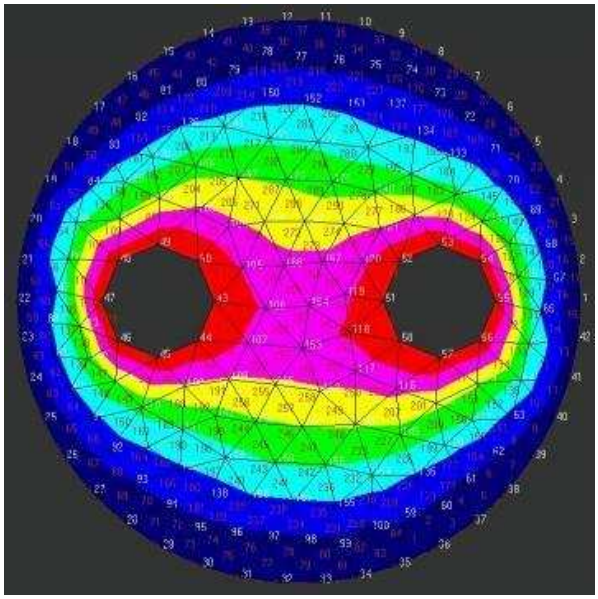


Abbildung 2.5: Beispiel eines FEM-Modells, das über den Einsatz der FEM-Methhode berechnet wurde[3]

2.2.3 Extract Transfer Load Workflows

Extract Transfer Load-Workflows(ETL-WF's) werden zur Modellierung, Orchestrierung und Ausführung von ETL-Prozessen eingesetzt [VsRM08]. Ein ETL-Prozess besteht aus mehreren ETL-Operationen. Diese ETL-Operationen lassen sich als Aktivitäten in dem Kontext eines Workflows darstellen. Ein solches Konzept basiert, z.B. auf die Erweiterung von WS-BPEL um Datenmanagement-Aktivitäten [RRS$^+$ny]. Diese Aktivitäten erlauben den direkten und nahtlosen Zugriff auf beliebige Datenressourcen. So ist es möglich, den Inhalt einer Datei durch eine Anfrage an das Dateisystem in die Workflow-Maschine zu laden oder den Inhalt einer Tabelle in einer relationalen Datenbank direkt abzufragen. Dieser Inhalt kann entweder als Resultat (z.B Retrieve Befehle wie SELECT) oder als eine Bestätigung der Ausführung von Definitions- oder Manipulationsanweisungen (z.B eine JOIN Anweisung) an die WF-Maschine geliefert werden.

Das SIMPL (SimTech-Information Management, Processes and Languages) Rahmenwerk wurde als Prototyp in Apache ODE für ETL-Operationen in Simulationsworkflows implementiert [RRS$^+$ny]. Das SIMPL Rahmenwerk erlaubt die Ausführung von ETL-Operation, wie z.B das Laden (Load/Retrieval), die Verknüpfung (JOIN), die Zusammenführung (MERGE) und die Vereinigung (UNION), von Daten, auf relationale Datenbanken, verschiedene Dateitypen (z.B Comma Separated Value (CSV)-Dateien) und andere heterogenen Datenressourcen in Simulationsworkflows [Wag11]. Die Ausführung von ETL-Workflows setzt meistens eine große Datenmenge voraus, deshalb ist das Datenaufkommen entsprechend groß. Man hat aber die Möglichkeit durch verschiedenen Techniken diese Daten von der Workflow-Maschine fernzuhalten. Die Parallelisierung von ETL-Workflows, zumindest bei

der Ausführung von ETL-Operationen auf gleichen Datensätzen bringt keine Verbesserung und ist daher sinnlos.

2.3 Wissenschaftliche Workflowmanagementsysteme und Simulationsrahmenwerke

In diesem Teilkapitel werden wir einige wissenschaftliche Workflowmanagementysteme (sWfMS) und Simulationsrahmenwerke (SR) vorstellen, die entweder im Rahmen dieser Arbeit verwendet werden oder in der Welt der rechnergestützten Simulationen bedeutend sind. Es wird zuerst die Architektur eines sWfMSs beschrieben. Danach werden wir einen Einblick in die sWFMSe SimTech, Kepler und Trident geben. Zum Schluss werden wir die SRe DUNE, ChemShell, Pandas und SIMPL kurz vorstellen. Dieses Kapitel basiert auf [BIS+06]. Andere Quellen werden explizit eingegeben.

Im Folgenden wird die Architektur eines sWfMs erläutert. Diese Architektur wurde in [KSK+11] auf einer konzeptionellen Basis vorgeschlagen und ist auf Abbildung 2.6 zu sehen. Die *Graphical User Interface* (GUI) stellt den ersten Bestandteil des sWfMS in dieser Architektur dar. Sie besteht im wesentlichen aus vier Komponenten:

1. *Service Catalog-Komponente:* Stellt dem Nutzer eine Liste von vordefinierten Modellierungsmuster zur Verfügung. Der Nutzer kann aus dieser Liste abstrakte Modellierungsmuster auswählen, um seine Aktivitäten zu definieren.
2. *Workflow Modeller-Komponente:* Unterstützt den Wissenschaftler bei der Spezifikation und der Modellierung seiner Workflows. Die Workflows können entweder graphisch oder direkt in der Workflow-Sprache erstellt werden.
3. *Monitor-Komponente:* Erlaubt dem Benutzer die Ausführung seiner Workflows zu überwachen.
4. *Result Display-Komponente:* Dient der Visualisierung der Zwischen- und Endergebnissen der Simulation.

Zur Realisierung des Deployments, der Ausführung, des Monitorings und anderer wichtigen Funktionalitäten von Workflows kommuniziert die GUI über fünf Schnittellen mit der *Runtime-Umgebung*. Die Runtime-Umgebung besteht aus folgenden Komponenten:

1. *Die Execution Engine* führt die Workflowinstanzen aus. Dazu instanziiert sie die definierte Workflow-Modelle aus der GUI, navigiert durch die Workflow-Graphen und ruft die Webservices, die die Aktivitäten eines Workflows implementieren, auf. Darüber hinaus werden verschiedene Ereignisse und Störungen von außen behandelt.
2. *Der Service Bus* kümmert sich hauptsächlich um den Aufruf von Diensten, die die Aktivitäten der Workflows implementieren.
3. Die *Resource Management-Komponente* verwaltet Metadaten bzw. Informationen zu den Ressourcen.

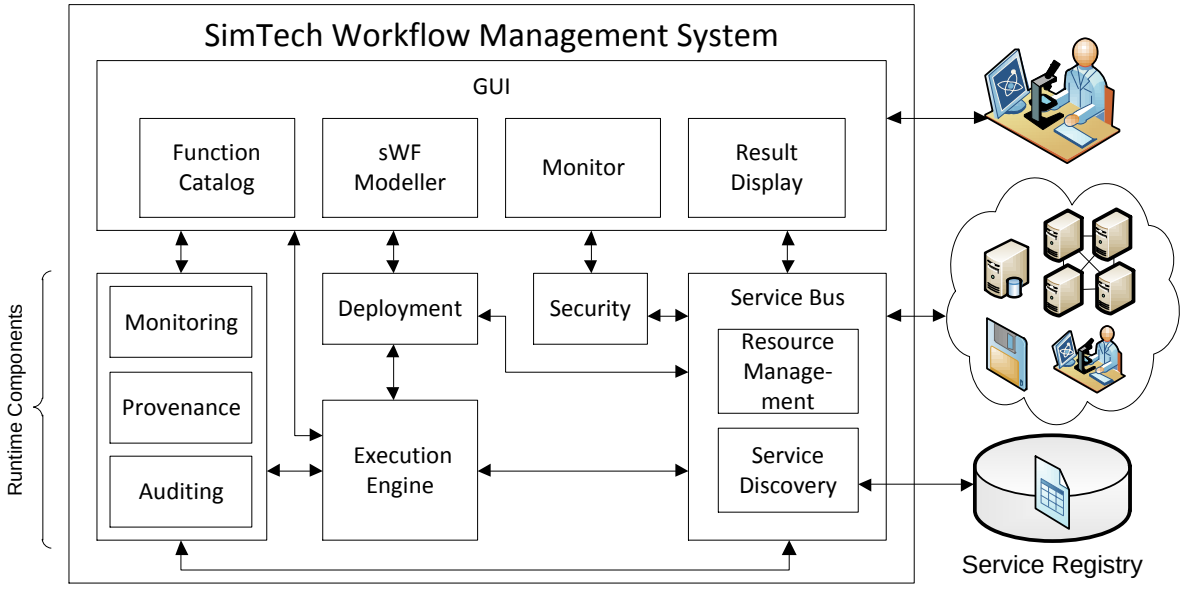


Abbildung 2.6: Architektur eines scientific workflow management systems[KSK+11]

4. Die *Service Discovery-Komponente* fragt nach Service Registries (z.B UDDI) nach, um Diensten mittels bestimmter Informationen (z.B Schnittstellen und semantische Annotationen) zu finden.
5. Die *Deployment-Komponente* macht die Workflow-Modellen aus der GUI der Runtime-Umgebung bekannt, indem sie die Modelle in der Engine installiert und die Workflows als Diensten veröffentlicht.
6. Die *Security-Komponente* kümmert sich um die lokale und ferne Sicherheitsrichtlinien, d.h schützt vor unautorisierten GUI und sWfMS Zugriffe.
7. Die *Auditing-Komponente* ist für die Aufzeichnung von Ereignissen, die während der Ausführung der Workflows ausgelöst werden, wie z.B die Ausführungszeit eines Workflows, zuständig.
8. Die *Monitoring-Komponente* verwendet diese Ereignisse, um den Status der Ausführung eines einzelnen Workflows abzufragen.
9. Die *Provenance-Komponente* zeichnet Ereignisse auf, die von der Auditing-Komponente nicht aufgezeichnet werden können.

SIMTECH

SimTech ist ein sWfMS, dass unter Mitwirkung von Mitarbeitern des Instituts für die Architektur von Anwendungssystemen(IAAS) der Universität Stuttgart im Rahmen der DFG Cluster Of Excellence Simulation Technology als Prototyp entwickelt wurde. Seine Funktionsweise basiert auf die im vorigen Abschnitt beschriebene Architektur [KSK+11]. SimTech wird im Bereich der Molekulardynamik, der modernen Mechanik, der

numerischen Mathematik, der Datenverwaltung in Simulationsworkflows und der interaktiven Visualisierung eingesetzt. Als Workflow-Sprache wird WS-BPEL (siehe Kapitel.2.2.1) eingesetzt und als Modellierungswerkzeug der Eclipse BPEL Designer verwendet. Auf Abbildung 2.7 ist die Modellierungsumgebung von SimTech zu sehen. Zur Modellierung von Workflows bietet"SimTech Modelling Perspective" den Workflow-Modeller und den Dienst-Katalog an. Darüber hinaus bietet dem Benutzer die Möglichkeit an, den Simulationsfortschritt über Monitoring-Komponente zu beobachten. "SimTech analysis perspective" erlaubt dem Benutzer, die Zwischen- und Endergebnisse der Simulation zu analysieren. Die Runtime-Umgebung verwendet OpenSource Software für alle Komponenten. Die Workflow-Maschine basiert auf die Apache Orchestration Director Engine(ODE) [4]. Zum Aufruf von Diensten wird Apache Axis2 [5] verwendet. Die Dienste Registrierung basiert auf jUDDI [6]. Zur Unterstützung der Ressourcen-Verwaltung im Service Bus, wurden eine Reihe von generischen Web Services entwickelt, um auf Datenquellen und wissenschaftliche Anwendungen zugreifen zu können. Zum Aufruf von wissenschaftlichen Anwendungen durch die WS-Schnittstellen, wird ein generischer Adapter eingesetzt. Zum Auditing hat man eine integrierte Datenbankumgebung entwickelt. Zum Speichern der Auditing-Informationen wird PostgreSQL[7] verwendet.

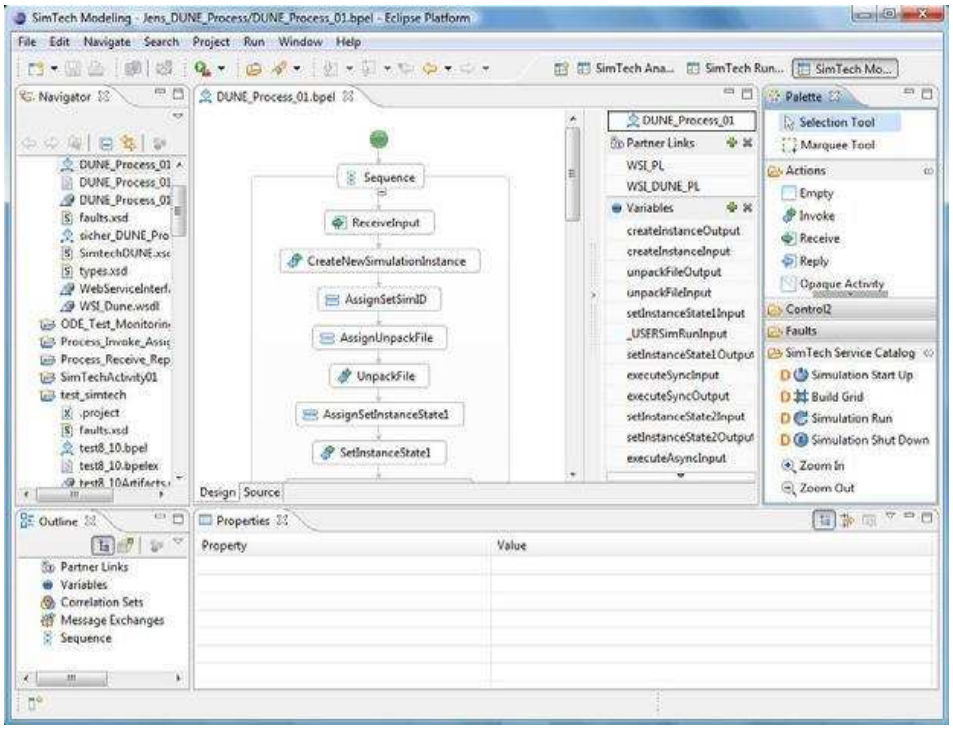


Abbildung 2.7: Die Modellierungsumgebung von Simtech [KSK+11]

[4]http://ode.apache.org/
[5]http://ode.apache.org/axis2/
[6]http://ode.apache.org/juddi/
[7]http://www.postgresql.de/

KEPLER

Kepler[8] ist ein sWfMS, das in verschiedenen wissenschaftlichen und Engineering-Bereiche eingesetzt werden kann. Zur Modellierung der Workflows wird eine Java basierte grafische Benutzeroberfläche verwendet. Auf Abbildung 2.8 ist diese grafische Benutzeroberfläche vom Kepler zu sehen. Der Wissenschaftler hat die Möglichkeit seine Workflows in einem Graphen basierte Struktur zu modellieren. Die Knoten in diesem Graphen repräsentieren Aktivitäten, die den Datenverlauf oder die Datenmodellierung darstellen. Die Kanten entsprechen dem Datenfluss zwischen den verschiedenen Aktivitäten. Die Kepler Bibliothek beinhaltet eine große Sammlung von solchen Aktivitäten, die dem Benutzer durch ein semantisches Suchsystem geliefert werden. Eine Aktivität kann von einem oder mehreren Rechner ausgeführt werden. Zur Ausführung einer Aktivität wird auf spezifische Datenspeicher dieser Rechner zugegriffen. Kepler erlaubt die direkte Ausführung der Workflows von der graphischen Benutzeroberflächen aus. Die Kepler Scientific Workflow (KSW)-Datei beinhaltet alle notwendige Informationen zur Ausführung eines Workflows auf ein beliebiges Kepler-System.

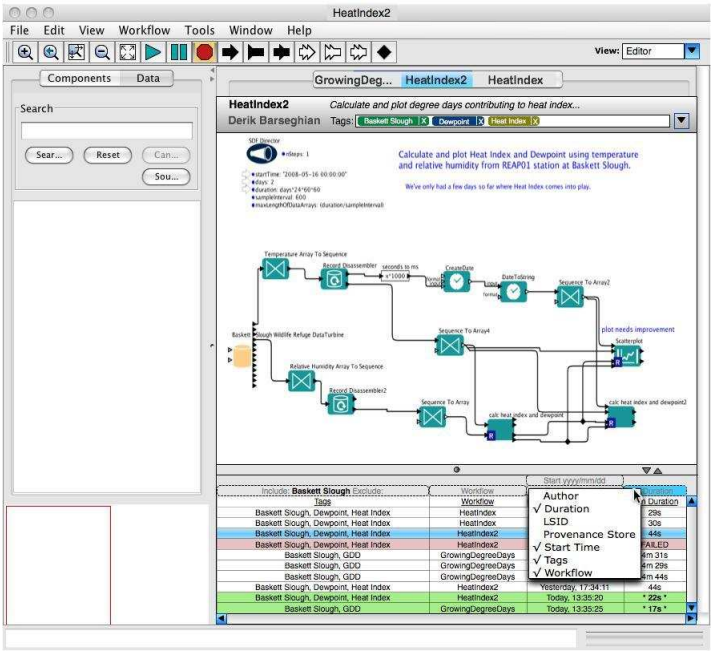


Abbildung 2.8: Die Kepler-Benutzeroberfläche [Mü10]

TRIDENT

Microsoft Trident [9] ist ein sWfMS, das auf Microsoft Windows Workflow Foundation

[8]http://www.kepler-project.org/
[9]http://research.microsoft.com/en-us/collaboration/tools/trident.aspx

[10] und das XML Format XOML basiert. Microsoft Trident wurde als ein Workbench für wissenschaftliche Workflows entwickelt [RJDN09]. Das Modellieren von Workflows erfolgt über einen Kontrollfluss orientierten Ansatz. Das Modellieren auf eine Datenfluss orientierten Basis wird über den Einsatz von Datencontainern unterstützt. Zur Modellierung von Workflows wird dem Benutzer eine GUI (Trident Workflow Composer) zur Verfügung gestellt sowie ein Monitorpool zur Überwachung der Ausführung der Workflows. Neben den Kontrollfluss orientierten Sprachelementen, wie z.B if-then Auswertungen, Schleifenstrukturen etc., bietet Microsoft Trident die Möglichkeit an, auf verschiedenen Datenressourcen zuzugreifen, wie z.B auf Dateisystemen und Datenbanken. Es gibt Aktivitäten, die die Erzeugung von Diagrammen sowie die Evaluierung von XPath-Ausdrucke auf XML-Dokumente, erlauben. Mit Trident, ist die Einbindung von Web Diensten ebenfalls möglich.

Zur Ausführung der Workflows gibt es zwei Möglichkeiten. Die erste Möglichkeit besteht darin, den Workflow in einer einfachen Workflow-Anwendung zu kompilieren und zu exportieren. Diese Workflow-Anwendung kann dann auf ein beliebiges System ausgeführt werden. Die zweite Möglichkeit besteht darin, den Workflow über die Trident-Laufzeit-Diensten auszuführen. Dazu kommt das Management Studio zur Administration, zum Monitoring und zur Untersuchung der Workflow-Instanzen zum Einsatz.

DUNE

Das DUNE-Rahmenwerk (Distributed and Unified Numerics Enviroment)[11] ist eine Software-Bibliothek, die eine Anzahl von Modulen zur Lösung von partiellen Differentialgleichungen anbietet. Zu diesem Zweck enthält das DUNE tollbox Implementierungen von Grid basierten Techniken, wie endliche Elementen und endliche Volumen. Darüber hinaus gibt es verschiedene Modulen zur Lösung linearer Gleichungen und iterativer Vektoren- und Matrixberechnungen. In der Bibliothek gibt es verschiedene Parser und Programme zum Lesen und Exportiren von Daten verschiedener Dateiformaten.

Der Kern von DUNE ist eine C++ Bibliothek, die als Quell-Code bereitgestellt wird. Das Erzeugen einer neuen DUNE-Simulationsinstanz erfolgt in mehreren Schritten. Es wird zunächst ein neues Modul mit einem Namen und einer Version erzeugt. Im zweiten Schritt werden in der DUNE-Bibliothek zusätzliche Module (z.B ein Modul zur Lösung linearer Gleichungen), die für die Ausführung der Simulation erforderlich sind, hinzugefügt. Wenn keine Module zur Implementierung der abstrakten Schnittstellen von DUNE vorhanden sind, dann müssen solche Module entwickelt werden. Weiterhin wird eine Routine zur Beschreibung der Simulationsschritte implementiert. In einem weiteren Schritt werden die Ergebnisdaten über die Definition und die Verwendung von Dateien exportiert. Schließlich werden die erforderliche DUNE-Module, die eigene Module und die Hauptroutine mit

[10] http://msdn.microsoft.com/de-de/library/aa480214.aspx
[11] http://www.dune-project.org/

dem GNU System [12] kompiliert. Nachdem diese Schritte durchgeführt wurden kann die DUNE-Simulation ausgeführt werden.

CHEMSHELL

ChemShell [13] ist ein Simulationsrahmenwerk dass ausschließlich zur Durchführung von Simulationen in der Chemie Industrie eingesetzt wird. Darunter zählen Simulationen aus dem Bereich der Quanten-Mechanik (QM), der Molekulare-Mechanik (MM) sowie der Hydro-Quanten- Mechanik/-Molekular-Dynamik (QM/MM).

ChemShell führt die Simulationen in Zusammenarbeit mit anderen externen Werkzeugen, die zur Lösung von speziellen Problemen (z.B Energie-Auswertungen) implementiert sind . ChenmShell sorgt für die Kommunikation dieser externen Werkzeugen, verwaltet die Simulationsdateien und verfügt integrierten Routinen zur Dateitransformation, Datenverarbeitung und Visualisierung. Darüber hinaus ist Chemshell in der Lage mit einfachen Berechnungen und numerischen Problemen umzugehen.

Das ChemShell-Rahmenwerk basiert auf die Script Sprache TCL (Tool Command Language)[14] . Es stellt dem Wissenschaftler einen interaktiven Shell zur Verfügung, dem er verschiedenen Befehle manuell eingeben kann. Diese Befehle werden anschließend zur Durchführung der Simulation interpretiert und ausgeführt. Eine Anzahl von solchen Befehlen werden von ChemShell bereitgestellt.

Es besteht die Möglichkeit die Simulation über die Verwendung eines Skriptes automatisch durchzuführen. Dieses Skript enthält vordefinierten Befehle, die einer nach dem anderen ausgeführt werden. Dieses Skript wird dem ChemShell als Eingabe-Parameter übergeben. Es startet, führt alle Befehle automatisch aus und schließt sich nach der Ausführung des letzten Befehls.

ChemShell basiert auf Simulationen, die über Skripten oder Benutzerinteraktionen durchgeführt werden. Deshalb muss die ChemShell-Umgebung nur einmal kompiliert werden. Aufgrund der Tatsache, dass die Simulationsaufbereitung vom TCL-Compiler durchgeführt wird ist die Kompilierung der Simulationsskripten nicht erforderlich. Wenn aber neue Befehle oder externe Module hinzugefügt werden, dann muss die ChemShell-Umgebung erneut kompiliert werden.

PANDAS

Pandas ist ein FEM-basiertes Rahmenwerk für Berechnungen bei porösen Medien [Dor11]. Poröse Medien Probleme treten in verschieden Engineeringbereichen auf. [15]. Im

[12]http://www.gnu.org/
[13]http://chemshell.org/
[14]http.//www.tcl.tl/
[15]http://www.mechbau.uni-stuttgart.de/pandas/index.php

allgemeinen schließen porösen Medien Modelle das interaktive Verhalten eines verformten Skeletts ein. Außerdem können Simulationen von thermischen sowie chemischen und elektrochemischen Phänomenen im Pandas-Rahmenwerk eingeschlossen werden. Man kann Pandas entweder auf einen Batch-Modus oder auf einen interaktiven Modus laufen lassen. Pandas bietet die Möglichkeit an die Simulation online zu visualisieren. Ähnlich wie beim DUNE werden zur Lösung von linearen Gleichungen und iterativer linearen Gleichungssystemen- und Matrizenberechnungen verschiedene Algorithmen eingesetzt.

SIMPL

Das SimTech-Information Management, Processes and Languages (SIMPL) ist ein Rahmenwerk zur generischen und erweiterbaren Datenmanagement-und Datenbereitstellungsabstraktion für Simulationsworkflows. SIMPL bietet die Möglichkeit an, direkt von einem Workflow aus, nahtlos auf beliebige Datenressourcen zuzugreifen [RRS$^+$ny]. Ein wichtiger Bestandteil dieses Rahmenwerks ist eine Erweiterung der Workflow Sprache BPEL, die es ermöglicht, neue BPEL-Aktivitäten zur Datenverwaltung in Simulationsworkflows zu definieren. Auf diese Art und Weise kann das Datenmanagement in Simulationsworkflows realisiert werden. in Kapitel 3 werden wir ausführlich über das SIMPL-Rahmenwerk diskutieren.

3 Das SIMPL-Rahmenwerk

In vielen Simulationsanwendungen müssen oftmals große, heterogene Datenmengen von einer Datenressourcen auf eine andere übertragen und transformiert werden. Das SIMPL-Rahmenwerk stellt eine Abstraktion zur Unterstützung der Datenverwaltung und -bereitstellung in solchen Simulationsanwendungen bereit. In diesem Kapitel werden zunächst die Architektur und die Funktionsweise dieses Rahmenwerks vorgestellt. Danach werden wir auf die Modellierung des Datenmanagements in Simulationsworkflows eingehen und die wichtigste Datenmanagement Aktivitäten des SIMPL-Rahmenwerks beschreiben. Zum Schluss werden wir einen Einblick in die Mechanismen, die das SIMPL-Rahmenwerk verwendet, um auf Datenressourcen zuzugreifen, geben. Dieses Kapitel basiert hauptsächlich auf [RRS+ny]. Andere Quellen werden explizit angegeben.

3.1 Architektur und Funktionsweise

SimTech-Information Management, Processes and Languages (SIMPL) ist ein generisches und erweiterbares Rahmenwerk für das Datenmanagement und die Datenbereitstellung in Simulationsworkflows. Auf der Modellierungsebene wird die Sprache WS-BPEL um zusätzliche Aktivitäten, die Datenmanagementoperationen gegen beliebigen Datenquellen direkt und nahtlos ausführen, erweitert. Um auf diese Datenquellen zuzugreifen, stellt SIMPL einheitlichen Zugriffsmechanismen und Metadaten bereit. Zur Modellierung der Workflows werden weiterhin Datenmanagementpatterns bereitgestellt, z.B für ETL-Operationen, die die Erstellung der Datenmanagementaktivitäten (DM-Aktivitäten) vereinfachen. Diese bereitgestellte Datenmanagementpatterns sowie die Datamanagementaktivitäten, die einheitliche Zugriffsmechanismen und die Metadaten erlauben die Definition des Datenmanagements für mehrere Applikationen im Bereich der Naturwissenschaften und der Ingenieurwissenschaften.

Abbildung 3.1 illustriert, wie das SIMPL-Rahmenwerk ein sWfMS (siehe Kapiel 2.3) erweitert, um eine Abstraktion für das Datenmanagement und die Datenbereitstellung in Simulationsworkflows zu ermöglichen. Zur besseren Übersicht werden alle Komponenten der sWfMSs-Architektur, die für SIMPL nicht relevant sind, weggelassen.

Die SIMPL-Architektur besteht aus folgenden Komponenten:

1. Die *SIMPL Core-Komponente* ist der Kern des SIMPL-Rahmenwerks. Sie ist im Service Bus des sWFMSs eingebettet und stellt für jede Art von Datenressourcen einheitlichen

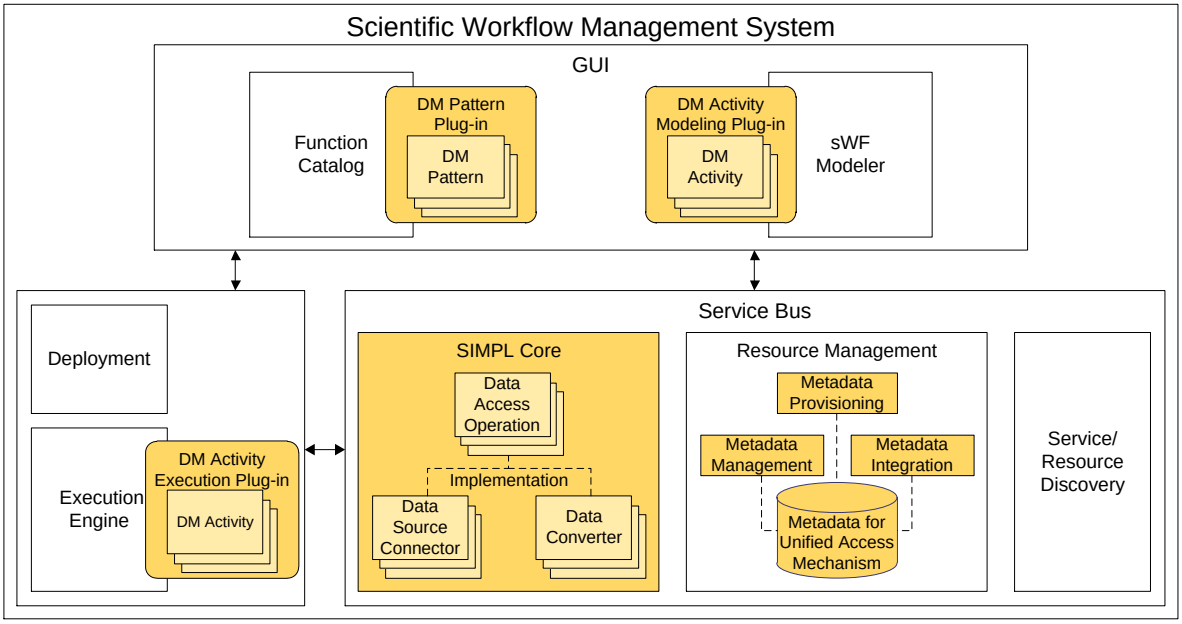


Abbildung 3.1: Integration des SIMPL-Rahmenwerks in die Architektur eines SWfMSs [RRS+ny]

und logischen Schnittstellen bereit. SIMpl-Core implementiert einheitlichen und logischen Schnittstellen auch für spezifische Datenressourcen (Connectoren und Converter) und agiert als ein Vermittler zwischen den Workflow-DM-Aktivitäten und den externen Datenressourcen. Hierbei leitet sie DM-Befehle, Ergebnisdaten und Benachrichtigungen über die erfolgreiche oder fehlgeschlagene Ausführung von DM-Befehlen weiter.

2. *Die Resource Management-Komponente* stellt Metadaten zur Beschreibung der Beziehungen zwischen den einheitlichen und logischen Schnittstellen des SIMPL-Cores und der konkreten Zugriffsmechanismen der Datenressourcen. Diese Metadaten werden in einer Datenbank abgespeichert.

3. In der *DM Activity Modelling Plug-in-Komponenente* befinden sich die DM-Aktivitäten. Diese DM-Aktivitäten können zur Modellierung des Datenmanagements in konkreten Simulationsworkflows verwendet werden.

4. *DM Activity Execution Plug-in-Komponente* befindet sich in der Workflow Execution Engine und führt die DM-Aktivitäten zur Laufzeit aus. Diese Aktivitäten können entweder direkt in den Simulationsworkflows integriert oder Bestandteil eines separaten ETL-Workflows (siehe Kapitel 2.2.3) sein.

5. Die *DM Pattern Plug-in-Komponente* der Funktions Catalogs-Komponente unterstützt den Workflow-Modellierer bei der Erstellung der notwendigen DM-Aktivitäten, indem sie abstrakte Datenmanagementpatterns für die Modellierung typischer DM-Operationen in Simulationsworkflows bereitstellt.

3.2 Modellierung des Datenmanagements in Simulationsworkflows

Die Business Process Execution Language (BPEL) (siehe Teilkapitel 2.2.1) ist ein de-facto Standard zur Definition und Ausführung von Geschäftsprozessen. In [ADR06] wird BPEL zur Modellierung und Ausführung von wissenschaftlichen Workflows vorgeschlagen. Um den direkten und nathtlosen Zugriff von einem Simulationworkflow aus auf externe Datenquellen zu ermöglichen, erweitert SIMPL BPEL um weitere Aktivitätstypen. Die o.g Aktivitäte heißen *Datenmanagement(DM)- Aktivitäten*. Die wichtigsten DM-Aktivitäten werden nachfolgend aufgelistet:

- IssueCommand-Aktivität
- RetrieveData-Aktivität
- WriteDataBack-Aktivität
- TransferData Aktivität

Das grundlegende Prinzip jeder DM-Aktivität ist die Ausführung eines DM-Befehls, z.B SQL-Ausdrücke oder Shell-Befehle, gegen eine bestimmte Datenressource. Unter einer *Datenressource* versteht man ein System, das Daten speichern, verwalten und/oder bereitstellen kann, z.B eine Datenbank oder ein Dateisystem. DM-Aktivitäten hat mindestens eine BPEL-Variable als einen Eingabeparameter. Diese Variable referenziert auf die Datenressource, gegen die der eingebettete DM-Befehl auszuführen ist. Wir nennen solche BPEL Variablen *Data Source Reference-Variablen*.

Die *Data Containe Referenz Variable* referenzieren auf einen Datencontainer. Ein *Datencontainer* (ist eine eindeutig identifizierbare Sammlung von Daten innerhalb einer Datenressource. In einer *Data Set-Variable* werden die in dem Prozesskontext eines Workflows geladenen Daten abgespeichert. Der Inhalt dieser Variablen wird von XML Schema-Definitionen spezifiziert. Für Tabellen-orientierte Daten, wie z.B Daten aus einer relationalen Datenbank oder aus einem CSV-basierten Datei, wird z.B eine *XML RowSet* Struktur verwendet.

An dieser Stelle werden wir auf die im ersten Abschnitt erwähnten DM-Aktivitätstypen eingehen. Die *IssueCommand-Aktivität* kann für Datenmanipulationen oder Datendefinitionen verwendet werden. Neben der Data Source Reference-Variable hat sie einen DM-Befehl als zusätzlichen Eingabeparameter. Dieser Befehl wird gegen die spezifizierte Datenressource ausgeführt. Teilweise werden auch etwas andere Parameter für diese Aktivität festgelegt, wie z.B DM-Befehl zur Ausführung eines Transformationsskriptes oder die Ziel-Datenressource eines Datencontainers.

Die *RetrieveData-Aktivität* hat ebenfalls einen DM-Befehl als Eingabeparameter. Über diesen Befehl werden Daten aus einer Datenressource extrahiert, z.B über einen SQL SELECT-Ausdruck oder einen Pfad zu einer Datei. Die Ergebnisdaten werden anschließend über die Workflow-Maschine in den Workflow-Kontext geladen. Ein zusätzlicher Eingabeparameter definiert eine Data Set-Variable, in der die Ergebnisdaten gespeichert werden.

Die *WriteDataBack-Aktivität* lädt Daten von dem Prozesskontext eines Workflows auf eine Datenressource. Diese Aktivität hat zwei Eingabeparameter. Einen Bezeichner für eine Data Set-Variable und einen für eine Data Container Reference-Variable. Sie speichert die Datenmenge der Data Set-Variable im Dateicontainer, auf den die Data Container Reference-Variable referenziert.

Die *TransferData-Aktivität* überträgt Daten von einem Datencontainern einer Datenressource auf einem anderen Datencontainer einer anderen Datenressourcen. Sie nimmt als Parameter zwei *Data Container Reference-Variablen*. Damit werden der Quell- und Ziel-Container bzw. die Datenressourcen, in denen sich die beiden befinden, eindeutig identifiziert. Teilweise werden etwas andere Parameter auch für diese Aktivität festgelegt, wie z.B die Ziel-Datenressource eines Datencontainers.

Data Container Reference-Variablen können zusätzlich als Parameter in DM-Befehlen der *IssueCommand*, der *RetrieveData* und der *TransferData* Aktivitäten, z.B in der FROM Klausel von einem SQL SELECT Ausdruck, verwendet werden. Dasselbe gilt für andere BPEL-Variablen, wie z.B String oder Integer Variablen, die für Vergleiche in Prädikaten benutzt werden können. In Bezug auf die Data Container Reference-Variable wird der logische Name des Datencontainers in dem Befehl eingetragen. Der SIMPL-Core sorgt für die Identifizierung dieses Datencontainers durch eine Abfrage an die Ressource Management-Komponente.

Das SIMPL-Rahmenwerk stellt über die DM Pattern Plug-in-Komponente der Function Catalog-Komponente (siehe Kapitel 3.1) eine Reihe von *Datenmanagementpatterns* bereit. Diese Datenmanagementpatterns decken wesentliche Datenbereitstellungsaufgaben für Simulationsworkflows ab. Der Workflow-Modellierer kann ausgewählte DM-Patterns als Hilfsmittel verwenden, um seine Datenmanagementoperationen auf einfachere Weise zu definieren. Das Beispiel der Join-Operation (siehe Kapitel 5.2), wird auf der Abbildung 3.2 illustriert. Der Nutzer muss in diesem Fall einige Parameterwerte setzen, z.B zwei Eingabe-Variablen, eine Ausgabe-Variable und eine Join-Bediengung, anstatt konkreten Datenmanagementoperationen zu definieren, welche die Verknüpfung von Datenmengen realisieren. Zur Identifikation und Unterscheidung von anderen Elmenten in einem DM-Befehl werden die Variablen mit umgebenden Nummerneichen markiert.

Die Abbildung dieses Patterns auf ein ausführbares Workflow-Fragment basiert auf der Anwendung von Regeln. Die zuvor spezifizierten Parameter der Patterns und Metadaten über diese Parameter entscheiden, ob eine Regel auf das Pattern angewendet werden kann oder nicht. Die Abbildung 3.2 zeigt die Transformationsregeln am Beispiel des parametrisierten Join-Patterns. Wenn die zwei Eingabe-Variablen und die Ausgabe-Variable des Join-Patterns auf Tabellen in derselben Datenbank verweisen, dann ist die Ausführung einer *Issue Command-Aktivität* ausreichend. Dieser Fall wird auf der Abbildung 3.2(b)) dargestellt. Falls hingegen die Ausgabe-Variable das Ergebnis der Join-Operation direkt im Workflow speichert, dann wird eine *RetrieveData-Aktivität* ausgeführt (siehe Abbildung 3.2 (c)). Wenn aber alle drei Variablen auf drei unterschiedlichen Datenressourcen referenzieren, z.B eine Comma-Separated Value (CSV)-Datei und eine Datenbanktabelle als Quelle und

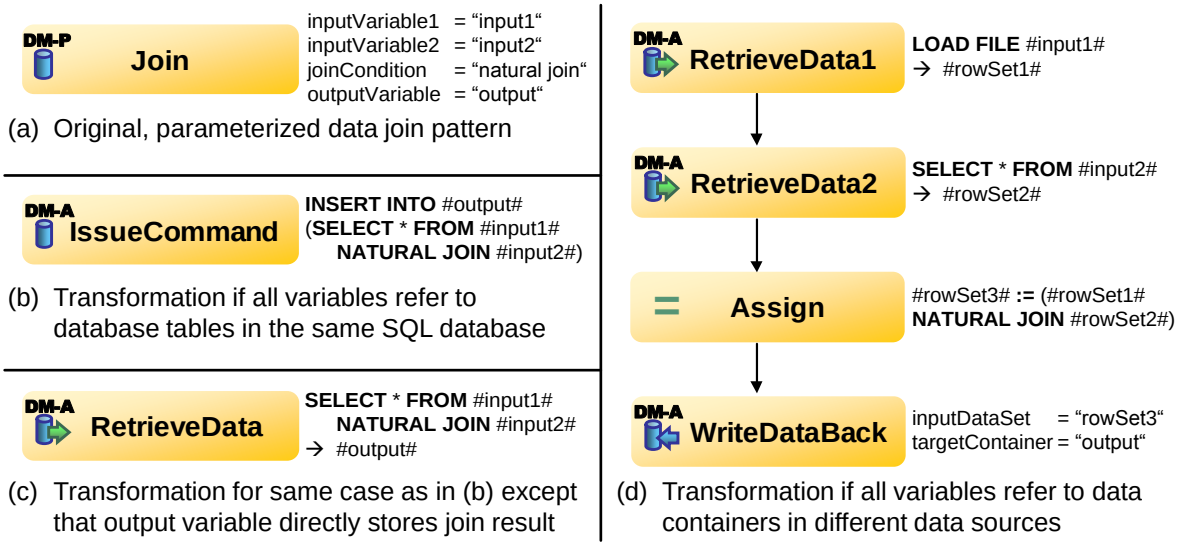


Abbildung 3.2: Das Join-Pattern und seine Transformation auf ausführbare Workflow-Fragmente[RRS+ny]

eine Datenkbanktabelle als Ziel, dann werden in diesem Fall zwei *RetrieveData-Aktivitäten* benötigt. Diese zwei *RetrieveData-Aktivitäten* laden die Daten aus den beiden CSV-Datein im Prozesskontext des Workflows. Anschließend verknüpft eine *BPEL Assign-Aktivität*, die beide Datenmengen und eine *WriteDataBack-Aktivität* speichert das Ergebnis der Join-Operation in der Zieltabelle der Datenbank ab (siehe Abbildung 3.2 (d)).

Diese Transformation kann erst festgelegt werden, wenn die involvierte Datenressourcen bereits bekannt sind. Im Falle einer statischen Einbindung der Datenressource kann die Transformation in der Deployment-Phase des Workflows stattfinden. Falls aber die Daten zur Laufzeit eingebunden werden sollen, dann wird jedes Pattern in ein einzelnes Workflow-Fragment konvertiert. Anschließend wird die Workflow-Fragment-Technologie eingesetzt, um diese Fragmente dynamisch in einem bereits ausgeführten Workflow zu integrieren [EUF09] [Hum11].

Neben dem Join-Pattern gibt es weitere Datenmanagementpatterns, die auf ausführbaren Workflow-Fragmente über die Anwendung spezifischer Regeln transformiert werden können. Darunter zählen u.a Patterns für die Datenübertragung von Datenressourcen auf anderen Datenressourcen oder Patterns, die weitere ETL-Operationen beschreiben. Auf das Thema Datenmanagmentpatterns und die Transformation der Patterns auf ausführbaren Workflow-Fragmente werden wir in den Kapiteln 5 bzw. 6 näher eingehen.

3.3 Metadaten zum Datenzugriff in heterogenen Umgebungen

Die Metadaten beschreiben die Datenabbildungen zwischen den vereinheitlichten Schnittstellen des SIMPL Core und den unter Umständen heterogenen Zugriffsmechanismen auf verschiedenen Datenressourcen. In der Resource Management-Komponente (siehe Abbildung 3.1) werden insgesamt vier Arten von Objekten registriert:

1. *Data Sources*
2. *Data Containers*
3. *Data Source Connectors*
4. *Data Converters*

Abbildung 3.3 zeigt die Klassifizierung der Metadaten sowie die Kardinalitäten der Beziehungen zwischen den einzelnen Metadaten-Klassen. Ein *Logical Source Name* wird genau eine konkrete Datenressource zugeordnet. Er agiert als Identifikator für diese Datenressource innerhalb des SIMPL-Rahmenwerks. Er kann als ein logischer Datenressource-Deskriptor innerhalb der Workflows verwendet werden. Ein logischer Datenressorce-Deskriptor ist entweder ein logischer Name oder ein Dokument, das einige funktionelle oder nicht funktionelle Anforderungen für den Zugriff auf eine Datenressource beschreibt. Die *Interface Description* beinhaltet Informationen über die Schnittstellen der Datenressourcen. Die *Security Entity*, z.B Benutzernamen und Password, ermöglichen den authentifizierten Zugriff auf die Datenressource. Die *Descriptin of Further Functional or not Functional Properties* beinhalten Eigenschaften der Datenressourcen, wie z.B die maximale Antwortzeit. Diese Beschreibungen werden in einem Late Binding verwendet. Das *Data Container Objekt* beschreibt die Datencontainer, die von den entsprechenden Datenressourcen verwaltet werden. Diesen ist ein *Logical Container Name* zugewiesen, der als eine Referenz für Datencontainer eingesetzt werden kann. Dieser Name wird auf einen konkreten *Local Container Identifier* abgebildet. Dadurch werden die Datencontainer innerhalb der Datenressourcen eindeutig identifiziert. Das *Data Source Connector Objekt* beschreibt ein Data Source Connector, das die Operationen der SIMPL Core-Komponente, die gegen bestimmte Datenressourcen ausführbar sind, implementiert. Die *Connector Properties Descriprtions* beschreiben die notwendige Eigenschaften einer Datenressource, die ein Connector heben muss, um eine Verbindung mit der Datenressource herstellen zu können. Die *Source Properties Descriprion* beschreiben die notwendige Eigenschaften eines Connectors, die eine Datenressource haben muss, um eine Verbindung mit dem Connector herstellen zu können. Diese Beschreibungen werden aufeinander abgeglichen, um den geeigneten Connector für eine gewisse Datenressource auszuwählen.

Über das *Data Format for Connector* wird ein konkretes Datenformat beschrieben. Auf dieser Art und Weise wird das Eingabe- bzw. Ausgabeformat, das der Connector unterstützt, dem *Data Converter Objekt* bekanntgegeben. Über das *Data Format for Converter* wird das Eingabe- bzw. Ausgabeformat, das der Converter unterstützt, dem *Data Source Connector Objekt* bekanntgegeben. Die Datenformate der beiden Objekten werden a miteinander verglichen, so dass Objekte, die das gleiche Datenformat unterstützen, gepaart

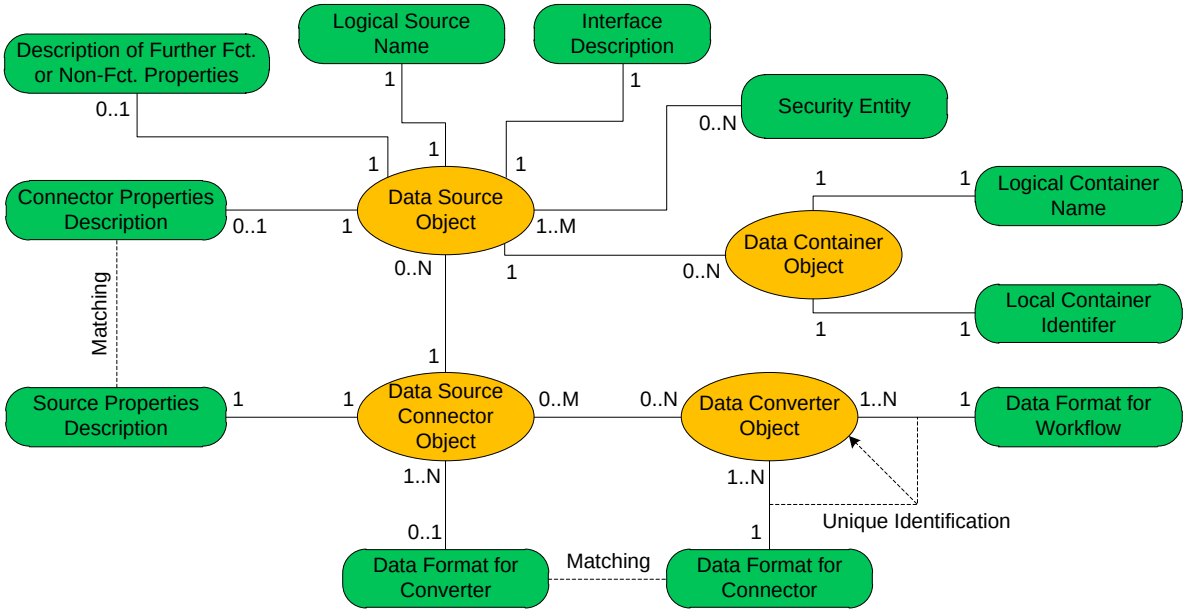


Abbildung 3.3: Klassifizierung der Metadaten zur Vereinheitlichung heterogener Datenzugriffsmechanismen [RRS+ny]

werden. Das *Data Format for Workflows* ist ebenfalls eine Datenformatbeschreibung, die das Data Converter Objekt braucht . Hierbei wird bekanntgegeben, welches Datenformat der Converter als Eingabe von einem Workflow erwartet und welches Datenformat der Converter als Ausgabe an den Workflow zurückschickt. Für jedes mögliche Datenformat-Paar gibt es maximal einen Converter.

3.4 Zugriff auf Datenressourcen mittels SIMPL

Um den Zugriff auf Datenressourcen über DM-Aktivitäten zu ermöglichen, muss der SIMPL-Core an konkreten Informationen bezüglich des Data Source Deskriptors, wie z.B die Beschreibung der Schnittstelle, den geeigneten Daten-Konnektor und den passenden Datenkonverter, kommen. Zusätzlich müssen Datencontainer in diesen Datenressourcen eindeutig identifiziert werden. Im Folgenden beschreiben wir wie die Service Bus-Komponenten miteinander interagieren, um den Datenzugriff zu ermöglichen. Dies wird auf Abbildung 3.4 dargestellt. Der an den SIMPL-Core gesendete logische Datenressourcen-Deskriptor (1) enthält spezifizierte Anforderungen für eine Datenressource (siehe Kapitel 3.2). Falls es unmittelbar auf eine konkrete Datenressource über einen Logical Name (siehe Kapitel 3.2) zugegriffen werden darf, dann überspringen wir die Schritte 2 bis 5.

Die spezifizierten Zugriffsanforderungen werden dann von der SIMPL Core-Komponente zu der Service/Ressource Discovery-Komponente geschickt(2). Die Service Discovery-Komponente schickt anschließend eine Anfrage an die Resource Management-Komponente.

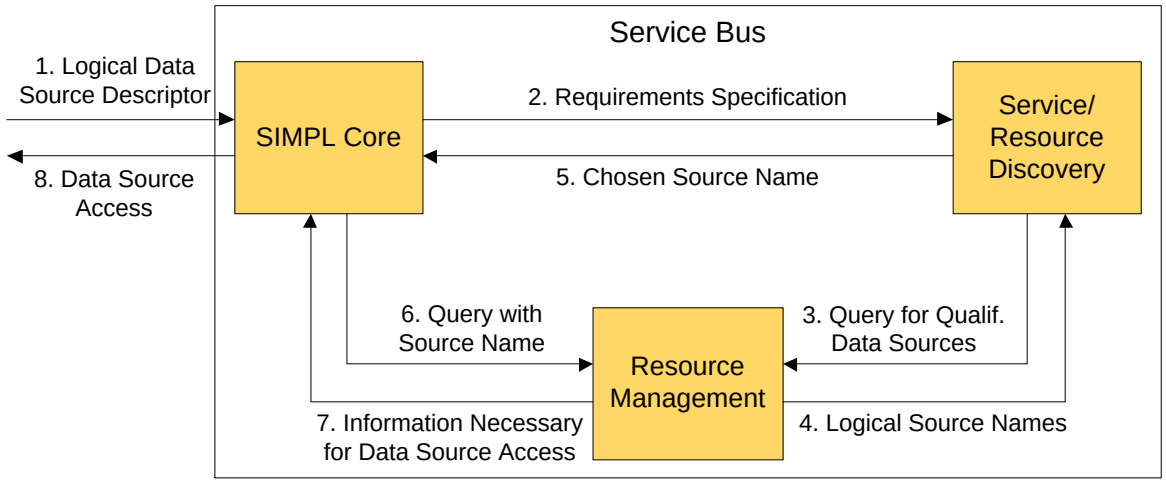


Abbildung 3.4: Interaktion der Service Bus Komponenten[RRS$^+$ny]

Letztere findet die passende Datenressourcen, die Zugriffsanforderungen erfüllen (3 und 4). Die Service Discovery-Komponent wählt basierend auf Kriterien in der Spezifikation der Zugriffsanforderungen, wählt eine Datenressource aus und schickt den logischen Namen der Datenressource zu der SIMPL Core-Komponente zurück(5). Die SIMPL Core-Komponente fragt in der Resource Management-Komponente nach allen notwendigen Informationen, die für den Zugriff auf die gewählte Datenressource notwendig sind, ab (6 und 7). Mit Hilfe dieser Informationen kann nun die SIMPL Core-Komponente auf die Datenressource zugreifen.

Im Rahmen dieses Kapitels wurde das SIMPL Rahmenwerk vorgestellt. Wie in diesem Kapitel beschrieben, stellt SIMPL u.a mit Hilfe von Datenmanagementpatterns eine generische und erweiterbare Datenmanagement- und Datenbereitstellungsabstraktion für Simulationsworkflows zur Verfügung. Es gibt aber auch eine Lücke, die noch geschlossen werden muss. Es müssen zusätzliche Abstraktionsmechanismen, die auf Datenmanagementpatterns basieren, entwickelt werden, um die Definition des Datenmanagements in Simulationsworkflows zu vereinfachen. Diese neue Datenmanagementpatterns werden anhand von konkreten Anwendungsszenarien, die im nächsten Kapitel vorgestellt werden, im Kapitel 5 identifiziert und herausarbeitet. Diese neue Datenmanagementpatterns werden anschließend im Kapitel 6 über die Festlegung geeigneter Abbildungsregeln auf ausführbare BPEL-DM Aktivitäten oder Service-Aufrufe abgebildet.

4 Simulationsworkflows

Im Rahmen dieses Kapitels werden die Anwendungsfälle vorgestellt, aus denen die Datenmanagementpatterns herausgearbeitet werden. Als erstes wird eine rechnergestützte Simulation einer chemischen Reaktion mit dem ChemShell-Rahmenwerk (siehe Kapitel 2.3) beschrieben. Als zweites Anwendungsszenario wird eine Simulation der Strukturänderungen eines Knochens unter bestimmten Belastungen auf den Knochen mittels des FEM-Simulationsrahmenwerks Pandas (siehe Kapitel 2.3) vorgestellt. Bei dem dritten Anwendungsszenario handelt es sich um die gleiche Simulation auf eine Multi-Domain-, Multi-Skelen- und Multi-Tool-Ebene (siehe Kapitel 2.2.2). Die Durchführung der Simulation erfolgt über eine Kopplung der Simulationsprogramme Pandas und Matlab. Die Vorstellung des Prozesses zur Reduktion eines komplexen mathematischen Modells, der von Simulationen durchgeführt werden kann, schließt dieses Kapitel ab. Das Teilkapitel 4.1 basiert auf [Mü10], das Teilkapitel 4.2 auf [RRS$^+$ny] und [RSM], das Kapitel 4.3 auf [Dor11] und das Teilkapitel zur Modellredektion auf [Rem11]. Andere Quellen werden explizit angegeben.

4.1 Chemische Reaktion mit Hilfe der Verwendung eines Katalysators

Mit dem ChemShell-Rahmenwerk können hybride quantenmechanische/molekularmechanische (QM/MM) Simulationen durchgeführt werden. Chemshell ist kein FEM-basiertes Rahmenwerk. In diesem Fall wird die Schrödinger Gleichung eingesetzt. Die Lösung der Schrödinger Gleichung ist typischerweise rechen-und datenintensiv und wird von anderen externen Werkzeugen übernommen. Chemshell stellt die Molekülstruktur als Eingabe für diese externen Werkzeuge bereit. Über einen iterativen Ansatz wird die Schrödinger Gleichung anhand der gegebenen Molekülstruktur von diesen externen Werkzeuge gelöst. Die Energie der Atome und die Kräfte zwischen denen lassen sich über die Ergebnisse der Simulation ableiten. Für weitere Details für QM/MM Kalkulationen, sei der interessierte Leser auf [P.So3] verwiesen.

Das folgende ChemShell Beispiel simuliert die Konversion des Glutamats in Methyl-Aspartat unter Verwendung des Enzyms Glutamat-Mutase als Katalysator. Das Beispiel illustriert eine typische hybride QM/MM Simulation. Aufgrund der Tatsache, dass die chemische Details dieser Reaktion nicht einer Bestandteil dieser Arbeit sind, wird dieses Beispiel auf eine vereinfachte Art und Weise präsentiert.

Auf der Abbildung 4.1 wird der Simulationsworkflow dargestellt. Die Simulation

startet mit dem Laden der benötigten Eingabedaten, wie z.b eine Protein-Datei, die aus der Protein-Datenbank[1] heruntergeladen werden kann. Zur Vorbereitung der Simula-

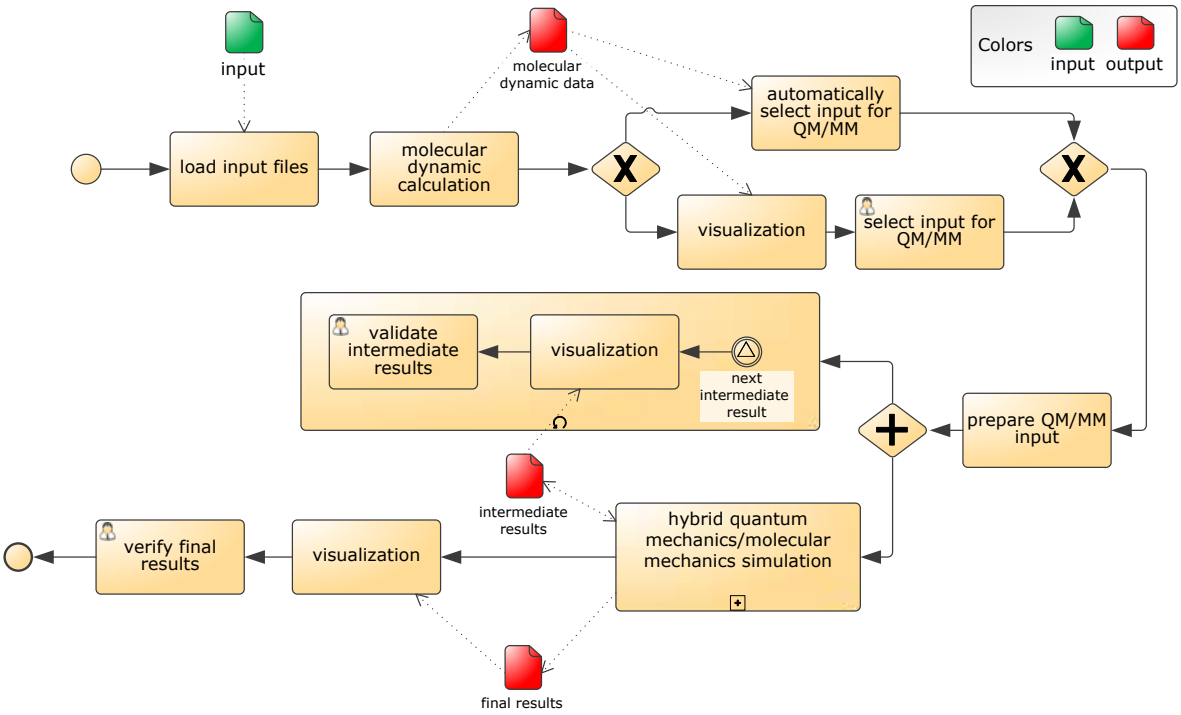


Abbildung 4.1: Simulationsworkflow einer chemischen-Reaktion mit Hilfe der Verwendung eines Katalysators [Mü10]

tion wird eine klassische *mulekur- dynamische Kalkulation* durchgeführt. Die Ergebnisse dieser Kalkulation werden in einer Datei abgespeichert. Diese Ergebnisse werden vom Wissenschaftler selbst oder von einer Aktivität, die dazu ein Programm, Werkzeug oder einen Service stößt, verwendet werden, um die Eingabedatendaten zur Durchführung der Simulation zu selektieren. Der Wissenschaftler hat an dieser Stelle auch die Möglichkeit über den interaktiven Modus auf die Simulationsvorbereitung einzugreifen. Er kann die Simulation abbrechen oder mit der Selektion der Eingabedaten weitermachen. Um dem Wissenschaftler dabei zu helfen, seine Entscheidung zu treffen, werden die Ergebnisse der molekular-dynamischen Kalkulation visualisiert.

Um die Eingabedaten für die *QM/MM-Simulation* vorzubereiten, werden zusätzliche Informationen auf Variablen, die vom Chem-Shell bereitgestellt werden, geladen. Die Durchführung der QM/MM Simulation wird von verschiedenen externen Werkzeugen übernommen, während ChemShell für die Bereitstellung der Variablen und die Speicherzuweisungen sorgt.

[1] http://www.rcsb.org/pdb/

Zur Durchführung der QM/MM-Simulation wird die Schrödinger-Gleichung gelöst. Die Berechnungen, die zu diesem Zweck durchgeführt werden, sind typischerweise sehr rechenintensiv. Daher kann die Simulation lange dauern. Weiterhin wächst die Anzahl der temporären Daten mit der Simulationszeit. Deshalb werden Zwischenergebnisse der Simulation sehr oft in verschiedenen Dateien abgespeichert. Diese Dateien müssen serialisiert und syntaktisch analysiert werden. Dieser Vorgang ist jedoch sehr aufwändig, weil diese Dateien sehr komplex und groß sind.

Ein Werkzeug visualisiert die Zwischenergebnisse, so dass der Wissenschaftler diese Zwischenergebnisse asynchron validieren kann. Wenn die Simulation beendet wird, werden die Endergebnisdaten auf verschiedenen Dateien geschrieben. Sie beinhalten alle Informationen, die zur Reproduktion der chemischen Reaktion notwendig sind. Der Wissenschaftler kann die chemische Reaktion *visualisieren*, indem er diese Dateien in einem geeigneten Visualisierungswerkzeug importiert. Dadurch kann der Wissenschaftler die Endergebnisse verifizieren.

4.2 Knochenmodellierung mit Pandas

Im Folgenden wird das zweite Anwendungsszenario vorgestellt. Es geht um einen Simulationsworkflow, der die Umformung einer Knochenstruktur unter bestimmten Belastungen auf den Knochen beschreibt. Das Pandas- Rahmenwerk (siehe Kapitel 2.4) simuliert diese Knochenstrukturänderungen auf der biomechanischen Ebene. Zu diesem Zweck wird die FEM-Methode (siehe Kapitel 2.2.2) eingesetzt. Abbildung 4.2 sind die Aktivitäten und die relevanten Eingabe- und Ausgabe-Daten des Simulationsworkflows zu sehen. Der Workflow ist in drei Phasen eingeteilt: die *preprocessing Phase*, die *Lösungsphase* und die *post-processing Phase.*

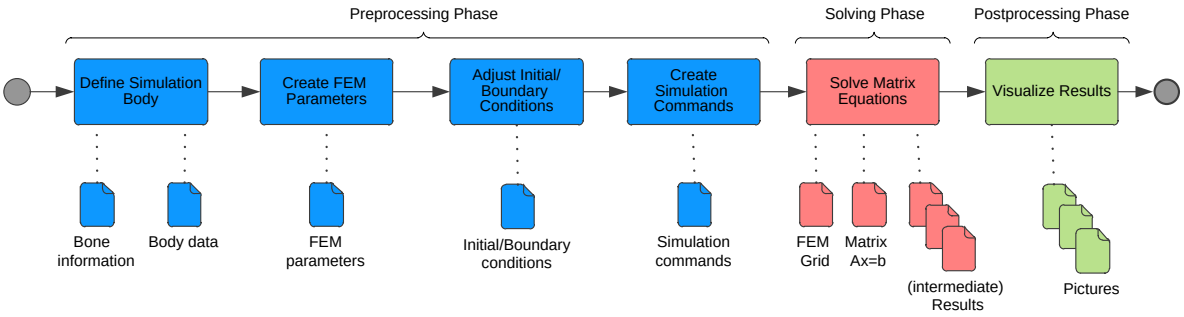


Abbildung 4.2: Der Pandas-Simulationsworkflow [RRS+ny]

In der *preprocessing Phase* startet er über die Aktivität *Define Simulation Body* mit dem Laden grundlegender Eingabe-Daten bezüglich des zu simulierenden Knochens aus verschieden Dateien. Beispiele solcher Informationen sind die geometrische Struktur des Knochens und die materielle Parameter des zu simulierenden Knochens. All diese Daten müssen in ein Format konvertiert werden, das während der Lösungsphase von Pandas

benötigt wird. Anschließend werden diese Daten in Dateien in der Pandas Umgebung zur Weiterverarbeitung abgespeichert. Die zweite Aktivität *Create FEM Parameter* extrahiert FEM-Parameter, z.B. Interpolationsfuktionen, aus einer unstrukturierten Datei, in der alle verfügbaren Parameter zusammengefasst werden. Danach stellt der Workflow die Anfangs-Bediengungen zur Konfiguration der Knochenstruktur der Simulation ein. Weiterhin werden die Randbediengungen definiert, z.B der zeitabhängiger Druck von außen auf den Knochen, der zur menschlichen Bewegung passend ist. Die relevante Eingabe-Daten dieser Aktivität werden aus verschiedenen Comma Seperated Value (CSV)-Dateien geladen. Die Wissenschaftler müssen nun Teilmengen dieser Daten selektieren und sie des benötigten Datenformats gemäß anpassen. Die letzte *preprocessing Aktivität* schreibt in einer Datei eine Menge von Simulationsbefehlen. Zum Beispiel wählt sie einen Matrixlöser aus und definiert die Diskretisierung der kontinuierlichen Simulationszeit in n Zeitschritte t_0 bis t_n.

In der Lösungsphase kommt Pandas in seine Simulationsschleife. Hierbei werden die Eingabe-Daten aus der *preprocessing Phase* verwendet. Damit werden die Matrix-Gleichungen der Simulation erzeugt. Zu jedem Zeitschritt t_i wird ein FEM-Gitter erzeugt oder angepasst, so dass die Basis zur Aufstellung der Matrix-Gleichungen $A_i \cdot x_i = b_i$ gebildet wird. Zur Generierung der Zwischen- und Endergebnisse werden diese Gleichungen anschließend gelöst. Ein FEM-Gitter beinhaltet eine große Anzahl von Gitterelementen. Diese Gitterelementen sowie die Matrizen A_i, die Vektoren x_i und b_i und die Zwischen- und Endergebnisse, die aus dem Vektor x_i abgeleitet werden, werden typischerweise im Hauptspeicher der Pandas-Software abgespeichert. Für einige Zeitschritte können diese Daten zur weiteren Behandlung in der *post-processing* Phase auch in einer Datenbank oder in Dateien gespeichert werden. *Die Lösungs-Phase* wird nach dem Zeitschritt t_n beendet. Abschließend speichert der Workflow die auf den Endvektor x_n basierenden Endergebnisse in einer CSV-Datei.

Zur Visualisierung aller zuvor gespeicherten Zwischen- und Endergebnisse sowie anderer relevanten Daten transformiert die Aktivität *Visualize Results* in der *Postprocessing Phase* all diese Daten in ein für Visualisierungswerkzeuge geeignetes Format. Zum Beispiel verknüpft sie für jeden relevanten Zeitschritt der Lösungs-Phase die FEM Gitterdaten und die Simulationsergebnisse, um Bilder zu kreieren, die diese Informationen kombinieren und überlagert darstellen. Damit hat der Wissenschaftler die Möglichkeit alle für ihn relevante Informationen graphisch zu analysieren.

4.3 Pandas-Matlab-Kopplung

Bei dem Anwendungsfall Pandas-Matlab-Kopplung geht es um eine Simulation, die auf einer Multi-Tool und Multi-Skalen Ebene stattfindet. Pandas berechnet die biomechanische Belastungen, die zur Änderungen der Struktur des Knochens führen, auf einer größeren Raumskala, während Matlab eine Berechnung der systembiologischen Komponenten der Kalziumbildung des Knochens auf zellularer Ebene und damit auf einer kleineren Raumskala durchführt. Weiterhin wird bei Matlab auf einer feineren Zeitskala gerechnet. Zur

Koordination und zum Datenaustausch zwischen den beiden Simulationsprogrammen und ihren Workflows wird ein Data-Manager-Workflow eingesetzt. Da die Implementierung der Pandas-Matlab Kopplung im Laufe dieser Arbeit noch überarbeitet wird, wird nicht auf die Implementierungsdetails eingegangen, sondern es folgt eine konzeptionelle Beschreibung dieses Anwendungsszenarios. Auf Abbildung 4.3 wird das Zusammenspiel zwischen den drei beteiligten Workflows dargestellt.

Der Workflow Pandas-Bone berechnet die Pandas-Simulation, wie im Kapitel 4.2 beschrieben wird. Der Workflow Matlab-Bone führt die Matlab Simulation aus. Der Workflow Daten-Manager agiert zwischen dem Pandas-Bone-und dem Matlab-Bone und ist dafür zuständig, den Workflow Matlab-Bone über den Workflow Pandas-Bone anzutriggern. Darüber hinaus, verwaltet er den Datenaustausch zwischen den beiden Simulationswerkzeugen. Im Rahmen dieser Arbeit ist es besonders interessant, wie der Datenaustausch zwischen den drei Workflows stattfindet.

Nachdem die *Preprocessing-Phase* des Pandas-Bone beendet wurde (siehe Unterkapitel 4.2) wird der Workflow Data-Manager über die Aktivität *Start Data-Manager* gestartet. Damit man eine Matlab-Simulation ausführen kann, muss zuerst eine neue Simulationsinstanz erzeugt und vorbereitet werden. Da aber die Simulation in diesem Fall von mehreren Simulationsinstanzen, die auf unterschiedlichen Matlab-Rechner liegen, gemeinsam ausgeführt wird, wird über die Aktivität *Prepare Matlab Instanzes* eine Schleife zur Vorbereitung der neuen Simulationsinstanzen gestartet. Beim ersten Durchlauf wird die erste Simulationsinstanz für Matlab erzeugt. Hierbei wird der Simulation eine neue ID vergeben und das dazugehörige Arbeitsverzeichnis erstellt. Anschließend wird der Siurce-Code in der neuen Simulationsinstanz kopiert und entpackt. Diese Simulationsdateien enthalten die M-Datei (diese Datei beinhaltet Matlab-Befehle) und weitere für die Simulation relevante Daten, wie z.B Initialisierungsdaten. Diese Schleife wird so lange durchlaufen, bis alle Simulationsinstanzen erzeugt und vorbereitet wurden. Als nächstes kommt der Workflow Pandas-Bone in seine *Lösungsphase* (siehe Unterkapitel 4.2) und berechnet eine bestimmte Anzahl an Simulationsschritten. Anschließend werden die Simulationsdaten in einer Datenbank abgespeichert. Dazu zählen die Gitter-Elemente und die darin enthaltenen Gauss-Punkte, die für die Ausführung der Matlab-Simulation relevant sind. Die Gauss-Punkte sind über die Gitter-Elemente verteilt. Abbildung 4.4 zeigt ein Beispiel, wie die Verteilung der Gauss-Punkte auf die einzelnen Gitter-Elemente eines zweidimensionalen Gitters aussehen kann.

Über die Aktivität *Prepare Matlab Bone* des Workflows Data-Manager werden die Matlab Berechnungen vorbereitetet. Bevor die Matlab Berechnung beginnt, findet eine Aufteilung des Simulationsraums und daher auch der Daten zwischen den einzelnen Matlab-Rechner statt. In der Aktivität *Prepare Matlab Bone* des Data-Manager Workflows werden aus der zuvor in der Datenbank abgespeicherten Berechnungsdaten, die Anzahl der Gitter-Elemente und der Gauss-Punkte je Element festgelegt. Die Gauss-Punkte (und implizit die Gitter-Elemente) werden nun auf die einzelnen Matlab-Rechner verteilt und berechnet. Auf diese Art und

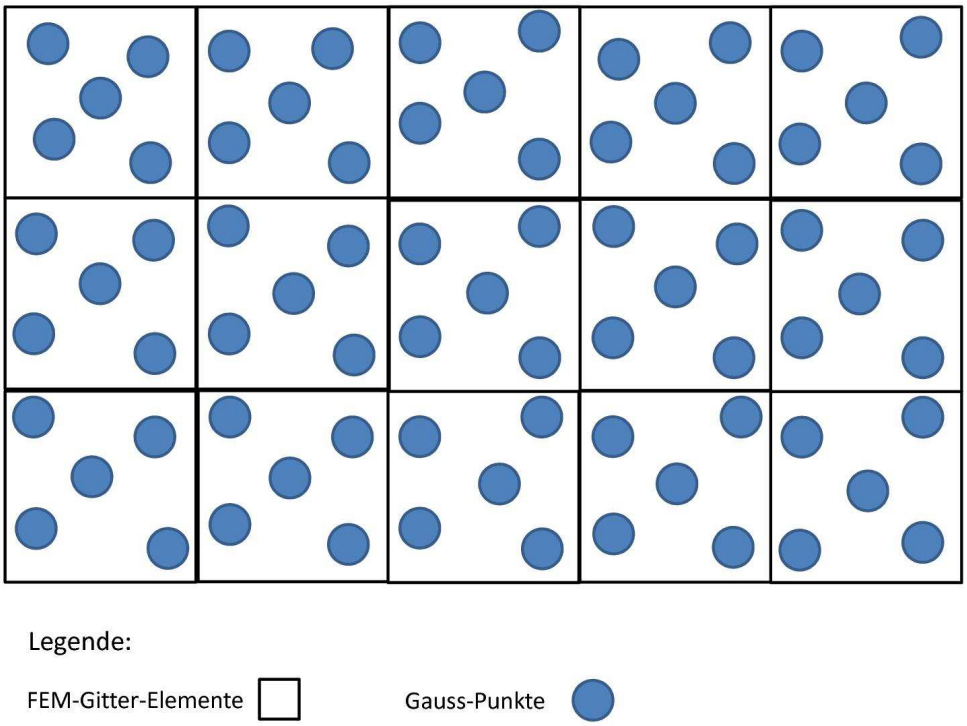


Abbildung 4.3: Beispielhafte Verteilung der Gauss-Punkte über die Gitter-Elemente

Weise findet eine Parallelisierung der Berechnungen der Gauss-Punkte statt.

Damit der Workflow Matlab-Bone die Simulationsberechnungen starten kann, müssen die Eingabe-Daten für die Matlab-Berechnungen für jede Matlab-Instanz vorbereitet werden. Hierbei werden zuerst über die Aktivität *Create Matlab Input-Data-File* die Eingabedateien erzeugt. Anschließend kopiert sie die aus der Pandas-Umgebung bereitgestellten Daten in eine CSV Datei. Über die Aktivität *Copy Input-Data-File to Matlabnode* werden diese Daten auf die jeweilige Matlab-Rechner kopiert. In dieser CSV-Datei der jeweiligen Matlab-Rechner stellt eine Zeile einen Gauss-Punkt und die Spalten dessen Variablenwerten dar. Das bedeutet, dass ein Gitter-Element, welches mehrere Gauss-Punkte enthält, sich über mehreren Zeilen der CSV-Datei erstreckt. Darüber hinaus werden zusätzliche Daten aus einer anderen Datei in einer zweiten CSV-Datei kopiert. Als nächstes wird über die Aktivität *Start Simulation* die Matlab-Simulation gestartet und auf die einzelnen Matlab-Ergebnisse gewartet. Über die Aktivität *Copy Output-Data-File from Matalab Node* werden die Ergebnisse jeder Matlab-Instanz auf zwei Ausgabedateien auf den Workflow-Rechner kopiert. Von einer davon (Beschreibung der Gauss-Punkte) wird der Inhalt dieser Datei mit dem der anderen Dateien der anderen Matlab-Instanzen gemerget und dann über die Aktivitäten *Insert Output-Data-File into DB* und *receive all Matlab-Bone* in die Pandas-Datenbank geladen.

Anschließend wird überprüft, indem man in Pandas-Bone zurückkehrt, ob die Simulation beendet werden soll oder nicht. Falls das n-te Simulationsschritt erreicht wurde oder die Ergebnisse der Simulation zufriedenstellend sind, wird die Simulation beendet.

Wenn die Simulation beendet wurde, wird der Data-Manager über den Aufruf der stop-Operation beendet. Nach dem Beenden des Data-Managers wird die Simulation abgeschlossen, indem Pandas-Bone in die *Post Processing-Phase* (sehe Unterkapitel 4.2) übergeht. Wenn die Simulation nicht beendet wurde dann wird weiter gerechnet.

4.4 Modellreduktion

Das letzte Anwendungsszenario, das im Rahmen dieser Arbeit untersucht wird, ist einer Prozess zur Reduktion eines mathematischen Modells, der von Simulationen durchgeführt werden kann. Dabei handelt es sich um die Reduktion eines Finite-Element-Modells eines elastischen Mehrkörpersystems. Eine Reduktion des mathematischen Modells bei elastischen Mehrkörpersysteme ist notwendig, da das mathematische Model aus unendlich abzählbaren gewöhnlichen Differentialgleichungen besteht. Der Prozess wird in drei Teilen aufgeteilt. Zuerst werden die Daten des zu reduzierenden Modells bereitgestellt, danach findet die Durchführung der Modellreduktion statt und zum Schluss werden die Ergebnisdaten ausgewertet, visualisiert und anschließend zur Weiterverarbeitung exportiert. Abbildung 4.5 zeigt den Ablauf der Modellreduktion.

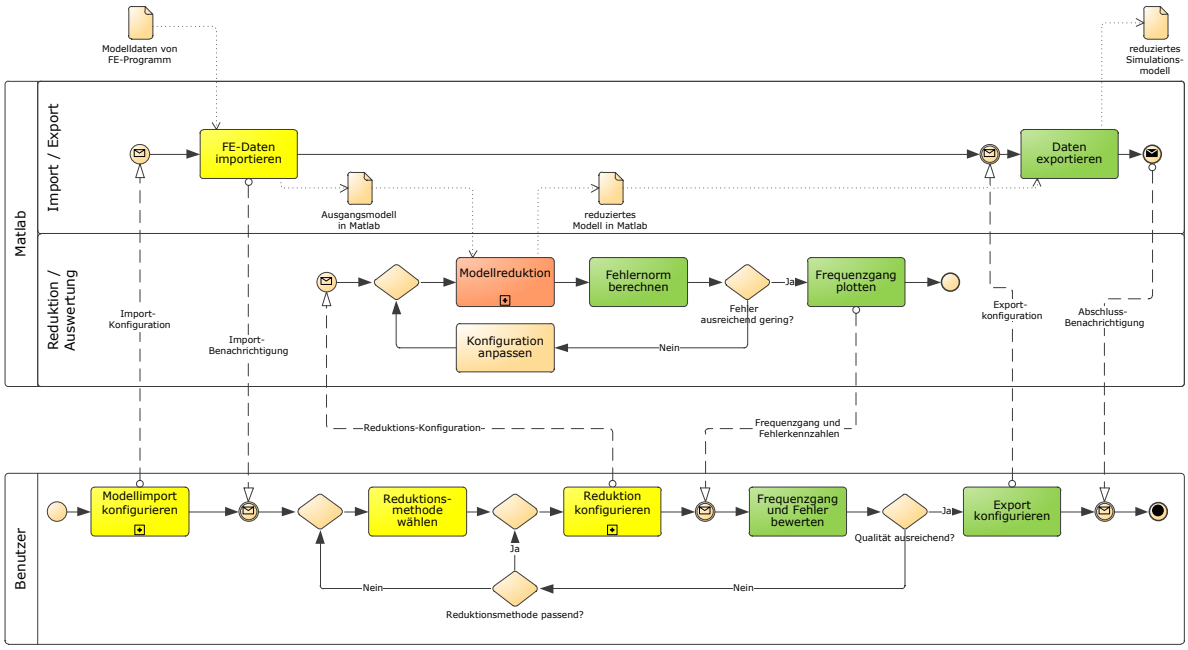


Abbildung 4.4: Der Prozess zur Modellreduktion [Rem11]

Der Simulationsablauf findet auf zwei Ebenen statt. Auf der *Benutzer-Ebene* wird die Simulation gestartet und gesteuert, während auf der *Matlab-Ebene* der Daten-Import und -Export sowie die Durchführung der Modellreduktion realisiert wird.

Damit mit der Simulation der Reduktion begonnen werden kann, müssen zuerst die notwendigen Daten des zu reduzierenden Modells in Matlab importiert werden. Bevor dies geschieht, müssen einige Parameter vom Benutzer konfiguriert werden. Dazu zählen die Parametrierung der Freiheitsgrade der Systemeingängen- und -ausgängen, die Definition der Dämpfung und der Starrkörperfreiheitsgrade und die Festlegung der Skalierung und der Ursprungsverschiebung. Zusätzlich muss eine Datenquelle ausgewählt werden, aus der die FE-Modelldaten gelesen werden. Diese Daten werden in den Matlab-Workspace importiert. Anschließend speichert die Aktivität *FE-Daten Importieren* all diese Daten, die den Input zur Durchführung der Simulation bilden, in einer Datei ab.

Nachdem die notwendigen Daten importiert wurden und der Benutzer die Reduktionsmethode ausgewählt und konfiguriert hat, kann mit der eigentlichen Durchführungsphase begonnen werden. Diese wird in einem iterativen Prozess eingebunden, der so lange durchläuft, bis die Ergebnisse des Reduktionsprozesses zufriedenstellend sind. Zur Durchführung des Reduktionsprozesses wird z.B die Methode der modalen Reduktion verwendet. Bei jedem Prozess-Schritt, müssen zuerst die Eigenvektoren, die den Unterraum v, auf den das zu reduzierende Modell projiziert wird, aufspannen, berechnet werden. Diese Eigenvektoren werden auch Eigenmoden genannt. Einige dieser Eigenvektoren werden nun ausgewählt, um die Projektionsmatrix aufzustellen und zu generieren. Abschließend lassen sich die Eigenfrequenzen des reduzierten Modells aus den Eigenvektoren berechnen. Für einige ausgewählte Prozess-Schritte werden die Ergebnisse der Modellreduktion in einer Datei abgespeichert. Anschließend werden die Daten aus dem Matlab-Workspace exportiert über die Aktivität *Daten exportieren* auf eine externen Datei kopiert.bzw in andere Datenformate transformiert werden.

Nachdem die Reduktion des Modells durchgeführt wurde, wird die Qualität der Ergebnisse über die Verwendung von Metriken automatisch überprüft. Sind die Ergebnisse nicht zufriedenstellend, dann wird der Prozess-Schritt, nachdem die ursprüngliche Konfiguration angepasst wurde, wiederholt. Diese Schleife wird solange durchlaufen, bis die Qualität der Ergebnisse ausreichend ist. Ist dies der Fall, dann übergeht man an die nächste Aktivität. Über die Visualisierung des Frequenzgangs und der Fehlerschätzung, kann auch der Benutzer an der Evaluierung der Qualität der Ergebnisse beteiligt sein. Ist der Benutzer mit der Qualität der Ergebnisse nicht zufrieden, dann wird der Simulationsschritt, unter Verwendung derselben oder einer anderen Methode der Modellreduktion, wiederholt. Ist die Qualität der Ergebnisse zufriedenstellend, dann werden diese zur Weiterverarbeitung in einer externen Datei abgespeichert.

Im Rahmen dieses Kapitels wurden die Anwendungsszenarien, aus denen Datenmanagementpatterns identifiziert und herausgearbeitet werden, vorgestellt. Während der Ausführung der Simulationen werden verschiedene großen Datenmengen, sowie heterogene Datenquellen und Datenformate involviert, wie z.B Datenbanken und CSV-Dateien.

Aufgrund dieser Tatsache würde eine manuelle Durchführung der Datenmanagementoperationen mit großer Wahrscheinlichkeit eine hohe Fehlerquote verursachen. Das SIMPL-Rahmenwerk (siehe Kapitel 3), bietet eine Lösung für dieses Problem an. Es ist aber schwer die Datenmangementoperationen für solche Simulationsanwendungen zu definieren. Eine generische und konsolidierte Datenmanagementabstraktion über die Entwicklung zusätzlicher Datenmanagementpatterns würde dieses Problem beseitigen.

5 Datenmanagementpatterns

In diesem Kapitel werden die im Rahmen dieser Arbeit entwickelten und verwendeten Datenmanagementpatterns (DMPs) vorgestellt. Zunächst wird im Kapitel 5.1 kurz auf die Pattern-Hierarchie eingegangen. Weiterhin werden wir grundlegenden Begriffe, die zum Verständnis dieses Kapitels benötigt werden, erklären und definieren. Im Kapitel 5.2 wird die erste Art von DMPs vorgestellt. Hierbei geht es um die Formalisierung und Beschreibung der ETL-Patterns/Operationen. Im Kapitel 5.2 wird es auf die zweite Art eingegangen. Das Data Transfer und Transformation Pattern und seine Container to Container Pattern, Data Split Pattern und Data Merge Pattern werden ebenfalls formalisiert und erläutert. Eine Formalisierung und Beschreibung des Data Iteration Patterns, der letzten Art der DMPs, schließt dieses Kapitel ab.

5.1 Datenmanagementpatterns und Pattern-Hierarchie

Unter einem DMP versteht man ein Entwurfsmuster für den Zugriff auf Datenressourcen und die Verarbeitung von Daten, die von diesen Datenressourcen verwaltet werden [BHH+10]. Im Rahmen dieser Arbeit werden solche DMPs auf einer abstrakten Ebene entwickelt, um die Definition des Datenmanagements in Simulationsworkflows zu vereinfachen und zu unterstützen. Für diese DMPs werden Datenmangagementoperationen definiert, die den Transfer, die Transformation und das Speichern von Datenmengen zwischen den Datenressourcen, die während der Ausführung eines bestimmten Simulationsworkflows involviert werden, ermöglichen. Zur Beschreibung und Klassifizierung der DMPs wurde eine Fünf Schichten-Hierarchie aufgebaut [RM11]. In diesem Dokument wird im Rahmen der Entwicklung von DMPs die Herleitung dieser Hierarchie erläutert. Auf der Abbildung 5.1 werden nicht alle Schichten angezeigt. Die Pattern-Hierarchie wird im Kapitel 6.1 vervollständigt. In der obersten Schicht der Pattern-Hierarchie befinden sich die *Anwendungs-Orientierte Patterns*. In dieser Schicht geht es um Patterns, die aus konkreten Simulationsanwendungen/Anwendungsszenarien identifiziert und extrahiert werden können. In der zweituntersten Schicht befinden sich die *ETL-Patterns/Operationen*, die typische DM-Operationen in Workflows darstellen [RRS+ny]. Über die ETL Patterns/-Operationen wird auf verschiedenen Datenressourcen zugegriffen, um das Extrahieren, die Übertragung, die Transformation und das Speichern von Datenmengen zu ermöglichen. Die ETL-Patterns sind für relationale Datenbankanwendungen bereits formalisiert und bekannt. Wir werden sie aber mit dem im Rahmen dieser Arbeit verwendeten Konzept entsprechend anpassen. In dieser Hierarchie gibt es weitere Schichten mit DMPs. Diese sind aber noch nicht formalisiert und werden daher als "unbekannte Patterns" betrachtet. In diesem Kapitel

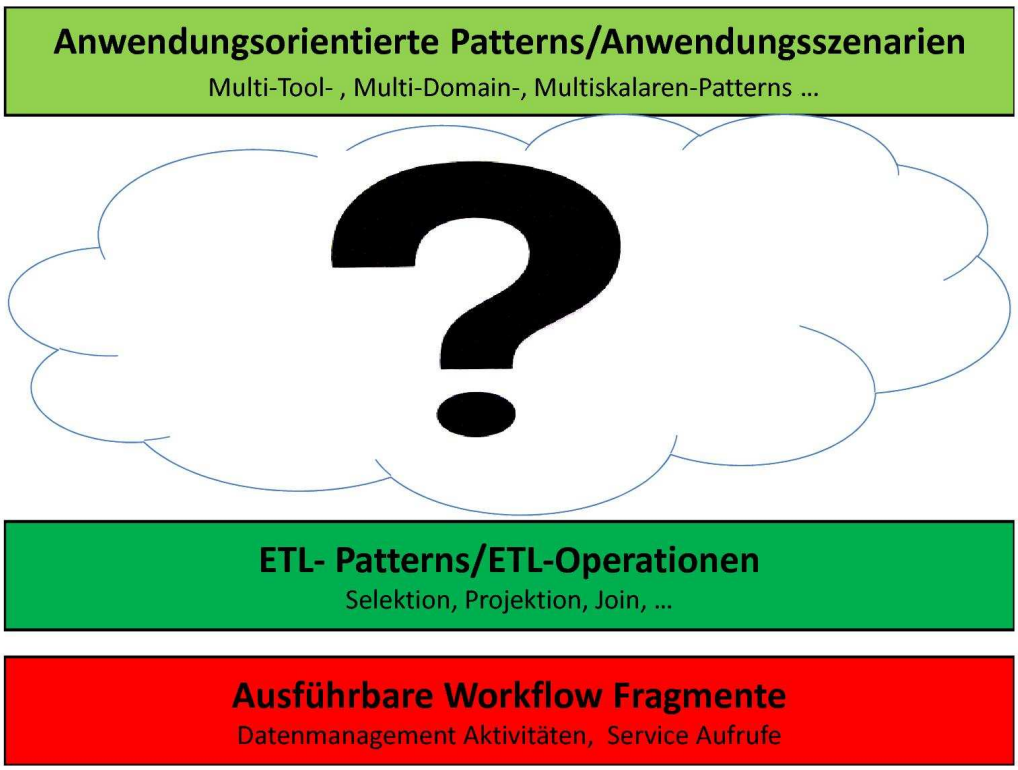


Abbildung 5.1: Die Pattern-Hierarchie vgl.[RM11]

werden wir diese Patterns formalisieren und erläutern. Auf der untersten Schicht befinden sich die Workflow-Fragmente, etwa BPEL-DM-Aktivitäten oder Service-Aufrufe, auf die die Datenmanagementpatterns abgebildet werden sollen. Wie das geschieht und welcher Mechanismus dabei verwendet wird, werden wir im Kapitel 6 erklären.

Bevor wir mit der Formalisierung und Erlaüterung der aus den Anwendungsszenarien identifizierten Patterns fortfahren, werden wir zunächst einige Begriffe, die für das Verständnis dieses Kapitels benötigt werden, erklären und definieren.

Unter einer *Datenressource* versteht man ein System, das Daten speichern, verwalten und/oder bereitstellen kann, z.B eine Datenbank oder ein Dateisystem vgl.[RRS$^+$ny]. Im Folgenden definieren wir abstrakt die Menge der Datenressourcen, die für unsere Patterns relevant sind:

$R = \{r_1, ..., r_m\}$ mit $i \in \{1, ..., m\}$ $m \geq 1$.

Eine *Datenquelle* ist eine Datenressource, die Daten bereitstellt, die der Nutzer bzw. der Klient extrahieren kann. Die Menge der Datenquellen bildet eine Teilmenge der Menge

der Datenressourcen. Im Folgenden definieren wir die Menge der Datenquellen:

$Q = \{q_1, ..., q_s\} \subseteq R$ mit $j \in \{1, ..., s\}$, $s \geq 1$ und $s \leq m$.

Eine *Datensenke* ist eine Datenressource, in der Daten abgespeichert werden können. Die Menge der Datensenken bildet die zweite Teilmenge der Datenressourcen. Die mathematische Definition der Menge aller Datensenken sieht wie folgt aus:

$P = \{p_1, ..., p_r\} \subseteq R$ mit $k \in \{1, ..., r\}$, $r \geq 1$ und $r \leq m$.

Eine Datenressource kann sowohl eine Datenquelle als auch eine Datensenke sein. Das bedeutet, dass die Schnittmenge der Mengen Q und P nicht unbedingt leer sein muss. Jede Datenressource (und damit auch jede Datenquelle und -senke) verwaltet eine Menge von Datencontainern. Ein *Datencontainer* (siehe Kapitel 3.2) ist eine eindeutig identifizierbare Sammlung von Daten innerhalb einer Datenressource. Im Rahmen dieser Arbeit unterscheiden wir zwischen persistenten und temporären Datencontainern. Einer *persistente Datencontainer* steht dauerhaft zu Verfügung. Zum Beispiel kann ein persistenter Datencontainer eine Tabelle einer Datenbank oder eine Datei in einem Dateisystem sein. Einer *temporäre Datencontainer* steht nur für einen bestimmten Zeitraum oder einen Gültigkeitsbereich zur Verfügung. Zum Beispiel kann ein temporärer Datencontainer eine BPEL Variable in einem Variablepool einer Workflow-Maschine oder eine JAVA Variable in einem Java Heap Size sein. Es folgt eine mathematische Definition der Menge aller Datencontainer aller Datenressourcen:

$C = \{c_1, ..., c_t\}$ mit $l \in \{1, ..., t\}$ und $t \geq 1$ und $t \geq m$.

An dieser Stelle möchten wir den Zusammengang zwischen den zuvor definierten Begriffen erläutern. Wir definieren $\forall\ c_l \in C$ die Abbildung:

$Cont : R \rightarrow P(C)$ mit $P(C) := \{U \mid U \subseteq C\}$ und $\emptyset \notin P(C)$.

Diese Abbildung ordnet jeder Datenressource $r_i \in R$ (und automatisch auch jeder Datenquelle $q_j \in Q$ und jeder Datensenke $p_k \in P$), Elemente der Potenzmenge von C zu, wobei $Cont(r_i)$ jder Datenressource r_i die Datenconrainer zuordnet, die diese Datenressource verwaltet.

Die Datencontainer aus C speichern Datenmengen. Unter einer Datenmenge S versteht man eine Teilmenge des kartesischen Produkts von bestimmten Wertebereichen. Diese Wertebereiche können dabei einfache Wertebereiche (z.B Integer) oder eine komplexe Datenstruktur sein (z.B ganze Datenmengen oder Graphen). Das kartesische Produkt wird aus dem Kreuzprodukt dieser Wertebereichen gebildet. Eine Datenmenge S ist damit wie folgt definiert:

$S \subseteq D_1 \times D_2 \times ... \times D_n$. [Cod70]

Die Wertebereichen D_1 bis D_n, aus denen das kartesische Produkt gebildet wird,

können den Attributen $A_1, ..., A_n$ von Datencontainern $\in C$ zugewiesen sein. Hierbei unterscheidet man beispielhaft zwei Fälle. Entweder beschreiben alle Attribute die Attributmenge eines einzelnen Datencontainers $\in C$ oder ein einzelnes Attribut beschreibt die ganze Attributmenge eines einzelnen Datencontainers $\in C$. Im ersten Fall wird also das kartesische Produkt innerhalb eines Datencontainers $\in C$ und im zweiten Fall zwischen mehreren Datencontainern $\in C$ gebildet. In der Regel ist eine solche Datenmenge S in einem oder mehreren Datencontainern $\in C$ gespeichert. Es kann aber vorkommen dass mehrere Datenmengen S in einem Datencontainern $\in C$ gespeichert sind.

Im Rahmen dieser Arbeit werden wir bei Beispielen ausschließlich mengenorientierten Datenstrukturen behandeln. Bei anderen Datenstrukturen wie objektorientierten-, baumartigen- oder hierarchischen Strukturen gehen wir nicht tiefer hinein. Das letztere gilt auch für alle im Rahmen dieser Arbeit vorgestellten Patterns. Für den *mengenorientierten* Fall definieren wir eine Datenmenge S wie folgt:

S=$\{s_1, s_2, ..., s_w\}$ mit a $\in$ 1,...,w und w $\geq$ 1.

Die Datenmenge S stellt im Beispiel der mengenorientierten Strukturen eine Menge von Tupeln dar. Die Tupel Datenelementen s_1 bis s_w sind jeweils ein Element aus dem kartesischen Produkt D_1 bis D_n.

5.2 ETL Patterns/Operationen

Nachfolgend definieren wir folgende ETL Patterns/Operationen:

- Selektion
- Projektion
- Datenformatkonvertierung
- Laden
- Vereinigung
- Verbund
- Aggregation
- Einfache Anreicherung
- Komplexe Anreicherung

Wenn eine Reihe von solchen ETL Patterns/Operationen durchgeführt wird, dann spricht man von einem *ETL-Prozess*. Die ETL Patterns/Operationen werden auf der Ebene der im Kapitel 5.1 definierten Datenmenge S beschrieben. Es wird also nachfolgend erläutert, was passiert, wenn einer der aufgelisteten ETL-Patterns/Operationen auf die Datenmenge S angewendet wird.

Selektion

Über die Anwendung der Selektion auf eine Datenmenge S wird eine Teilmenge $T \subseteq S$ selektiert.

Für den Fall, dass wir beispielhaft mengenorientierte Daten betrachten, sieht die Datenmenge S wie folgt aus:

$S = \{s_1, ..., s_w\}$ mit a $\in$ {1 ,..., w} und $w \geq 1$ mit einzelnen identifizierbaren Elementen s_a.

Die Menge $T \subseteq S$ ergibt sich wie folgt:

T=$\{s_{u_1}$,..., $s_{u_p}\}$ mit $v \in \{1, ..., p\}$, $p \geq 1$ und $p < w$, wobei jedes $s_{u_v} \in T$ einem der $s_a \in S$ entsprechen muss.

Über die Selektion erfolgt eine tupelweise Selektion von Daten aus der Datenmenge S. Die selektierten Tupel $\in S$ stellen dann die Ergebnismenge T dar. Die hier beschriebenen Datenmengen S wird bei der Definition der übrigen ETL Patterns/Operationen verwendet. Der Leser ist dann auf diesen Abschnitt verwiesen.

Ein Anwendungsbeispiel dieses Patterns findet man bei der Pandas-Simulation (siehe Kapitel 4.2). Die Selektion der Daten der geometrischen Struktur des zu simulierenden Knochens aus einer CSV-Datei, die diese Daten enthält stellt dieses Anwendungsbeispiel dar.

Projektion

Gegeben sei einer Datencontainer $c_l \in C$, der eine Datenmenge S_l verwaltet. Über die Anwendung der Projektion werden zunächst Attribute aus der ursprünglichen Attributmenge eines Datencontainers $c_l \in C$ ausgewählt. Daraus wird die Menge T_l gebildet. Die Elemente der Menge T_l entsprechen den Werten der projizierten Attribute.

Für den Fall, dass wir beispielhaft mengenorientierte Daten betrachten, sieht die Projektion wie folgt aus.

$\pi_{A_1, A_2, ..., A_k}(S_l) = \{ p \mid \exists\, t \in S_l \text{: } p =< t[A_1], t[A_2], ..., t[A_k] > \}$ [Cod70]

Über die Projektion werden also Spalten (Attribute) $A_1, A_2, ..., A_k$ aus der Attributmenge $A_1, A_2, ..., A_n$ eines Datencontainers $c_l \in C$ ausgewählt. Als Ergebnis ergibt sich das Element p $\in S_l$. Das Element p entspricht einem Tupel t $\in S_l$, wobei jedes Tupel dem Wert eines Attributes aus $A_1, A_2, ..., A_k$ entspricht.

Einen Anwendungsfall der Projektion findet man bei dem Anwendungszenario der Pandas-Matlab Kopplung (siehe Kapitel 4.2). Variablen der Gauss-Punkte (die Variablen

entsprechen Spalten des Datencontainers, in dem die Gauss-Punkte gespeichert sind), die für die Durchführung der Simulation auf der Matlab-Ebene als Eingabedaten nicht relevant sind, werden über eine Projektion gefiltert, so dass nur die relevanten Variablen der Gauss-Punkte übrig bleiben.

Datenformatkonvertierung

Die Datenformatkonvertierung transformiert eine Datenmenge S aus einem bestimmten Datenformat auf eine Datenmenge T eines anderen Datenformats. Zu diesem Zweck wird eine Konvertierungsoperation verwendet.

Ein Anwendungsbeispiel dieses Patterns ist die Konvertierung der Geometrie-Daten in einem Datenformat, das für die Lösungsphase des Pandas-Workflows (siehe Kapitel 4.2) benötigt wird.

Laden

Dieses Pattern verwenden wir, um eine Datenmenge S in einem oder mehreren Zielcontainer $\in C$ abzuspeichern.

Das Laden der Endergebnisse der Pandas-Simulation von der Pandas-Umgebung auf eine CSV-Datei (siehe Kapitel 4.2) stellt einen Anwendungsfall dieses Patterns dar.

Vereinigung (Union)

Über die Vereinigung werden mehreren Datenmengen S in eine Menge T vereinigt. Eine mathematische Definition der Vereinigung sieht wie folgt aus:

$T = S_1 \cup S_2 \cup ... \cup S_n$

Für den mengenorientierten Fall definieren wir die Vereinigung wie folgt:

$T = S_1 \cup S_2 \cup ... \cup S_n = \{t \mid t \in S_1 \vee t \in S_2 \vee ... \vee t \in S_n\}$. [Cod70]

Die Datenmengen S_1 bis S_n stellen hier mehrere Mengen von Tupeln dar. Die neue Menge, die sich aus der Vereinigung der Tupeln t aus diesen Mengen ergibt, ist dann die Ergebnismenge T. Zu beachten ist, dass die Vereinigung nur anwendbar ist, wenn die Datenmengen vereinigungsverträglich sind. Das bedeutet, dass sie die gleiche Anzahl von Attributen haben müssen und die Datentypen der Attribute gleich oder vergleichbar sein müssen [Cod70].

Ein Anwendungsbeispiel dieses Patterns findet man in dem Szenario der Pandas-Matlab Kopplung (siehe Kapitel 4.3). Die Ergebnisdaten jeder Matlab-Instanz werden jeweils in einer Ausgabedatei gespeichert. Diese einzelnen Ausgabedateien werden anschließend in einer Ergebnismenge vereinigt und in einer einzigen Ausgabedatei abgespeichert.

Erweitertes kartesische Produkt

Nachfolgend wird das erweiterte kartesische Produkt auch für den mengenorienterten Fall definiert:

$S_1 \times S_2 := \{t \mid \exists\ x \in S_1, y \in S_2 : (t = x \mid y)\}$ mit $x \mid y =< x_1, ..., x_r, y_1, ..., y_s >$ [Cod70]

Über die Anwendung des erweiterten kartesischen Produkt werden die Paare (x,y) mit x $\in S_1$ und y $\in S_2$ verknüpft, wobei x_i bzw. y_j z.B der Wert des Tupels x bzw.des Tupels y im i-ten bzw. j-ten Attribut ist. Betrachtet man das Beispiel der mengenorientierten Strukturen sind die Elemente x und y Tupel $\in S_1$ bzw $\in S_2$.

Verbund (Join)

Der Verbund wendet zuerst das erweiterte kartesische Produkt auf eine Datenmenge S an. Anschließend wird aus der Ergebnismenge eine Teilmenge T selektiert.

Im Falle des Beispiels der mengenorientierten Strukturen werden zuerst alle Tupel der Datenmenge S über das erweiterte kartesische Produkt miteinander verknüpft. Anschließend wird auf die Menge, die sich aus der Verknüpfung dieser Tupel ergibt, die Selektionsoperation angewendet. Die selektierten Tupel, nämlich die Teilmenge T, ist das Ergebnis der Verbund-Operation.

Dieses Pattern kommt beispielhaft bei der *Postprossecing* Phase der Pandas-Simulation zum Einsatz (siehe Kapitel 4.2). Hierbei werden die FEM Gitterdaten und die Simulationsergebnisse, die in zwei unterschiedlichen CSV-Dateien gespeichert sind, für jeden relevanten Zeitschritt der Lösungs-Phase verknüpft, um Bilder zu kreieren, die diese Informationen kombinieren und überlagert darstellen.

Aggregation

Gegeben sei ein Datencontainer $c_l \in C$, der eine Datenmenge S_l verwaltet. Die Aggregation wählt zunächst ein Attribut aus der ursprünglichen Attributmenge des Datencontainers $c_l \in C$ aus. Auf die Elemente der Teilmenge $T_l \in S_l$, die als Wert das projizierte Attribute besitzen, wird eine Aggregationsoperation angewendet. Anschließend werden alle Attributwerte, die die gleiche Eigenschaft erweisen, in einem einzigen Wert zusammengefasst.

Wenn man beispielhaft mengenorientierte Datenstrukturen betrachtet, wird über die Anwendung der Aggregation ein Attribut eines Datencontainers $c_l \in C$ selektiert. Auf die Tupel, die den Werten des selektierten Attributes entsprechen, wird eine Aggregationsoperation angewendet. Anschließend werden alle Tupel , die den gleichen Attributwert erweisen, in einem einzigen Wert zusammengefasst und nach einem anderen Attribut gruppiert.

Beispiele von Aggregationsoperationen sind: [1]

- COUNT: Gibt die Anzahl der Werte in einer Spalte an.
- SUM: Summe der Werte in einer Spalte
- AVG: Mittelwerte einer Spalte
- Max: Größter Wert der Spalte
- MIN: Kleinster Wert der Spalte

Die Festlegung der Anzahl der Gauss-Punkte pro Gitter-Element (siehe Kapitel 4.3) stellt ein Anwendungsfall dieser Operation dar. Bevor die Gauss-Punkte auf die einzelnen Matlab-Instanzen verteilt werden, muss zuerst festgestellt werden, wie viele Gauss-Punkte ein Gitter-Element besitzt. Das erfolgt über die Anwendung der Aggregation und der Operation COUNT. Anschließend werden die Gauss-Punkte nach deren Gitter-Elementen gruppiert.

Einfache Anreicherung

Bei der einfachen Anreicherung geht es um einen Neudatengewinn innerhalb eines Datencontainers. Diese Operation beschreiben wir anhand eines mengenorientierten Beispiels. Es werden zuerst Attribute von der Attributmenge eines Datencontainers $c_l \in C$, in dem die Datenmenge S_l gespeichert ist, ausgewählt. Anschließend wird auf Tupel $\in S_l$, eine Operation, wie z.B Multiplikation oder Addition, angewendet. Die aus der Anwendung dieser Operation gewonnenen Daten werden in der Datenmenge S_l integriert und in einem neuen Attribut abspeichert.

Komplexe Anreicherung

Bei der komplexen Anreicherung geht es, wie bei der einfachen Anreicherung, um einen Neudatengewinn. Die komplexe Anreicherung findet aber zwischen mehreren Datencontainern statt und außerdem können hier zusätzlich ETL-Operationen wie Join, Union, Merge etc. zum Einsatz kommen.

5.3 Data Transfer and Transformation Pattern

Das *Data Transfer and Transformation Pattern (DTTP)* bildet die zweite Art der im Rahmen dieser Arbeit vorgestellten Datamanagementpatterns. Sowohl dieses Pattern als auch

[1] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Gruppenfunktionen

seine Unterklassen, die in den nächsten Kapiteln vorgestellt werden, werden wir auf der Ebene der Datencontainer (siehe Kapitel 5.1) beschreiben. Wir werden also erläutern, wie Daten zwischen einzelnen Datencontainern $\in C$ übertragen und transformiert werden, wenn die im Rahmen dieses Kapitels vorgestellten Patterns gegen diese Datencontainer durchgeführt werden. Auf der Abbildung 5.2 wird das DTTP dargestellt: Um das DTTP

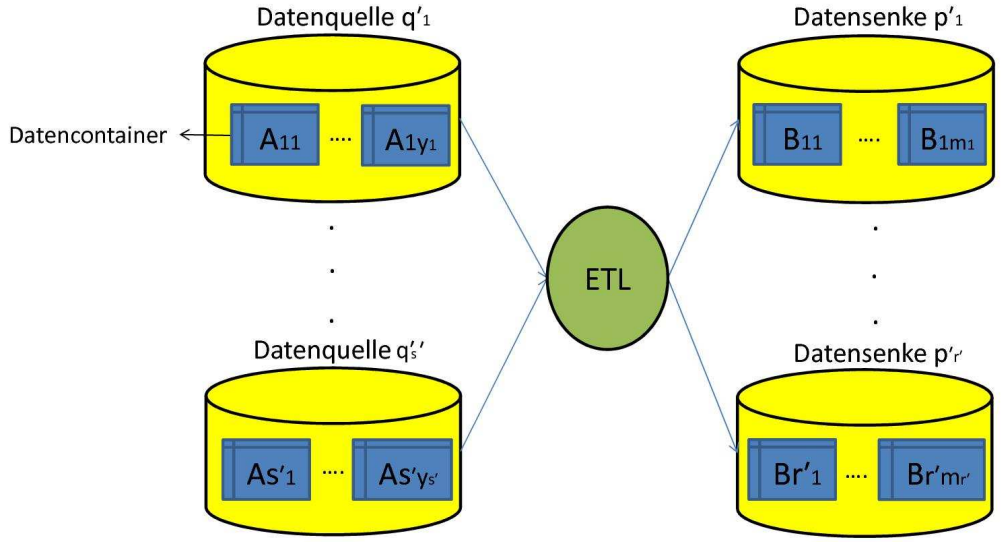


Abbildung 5.2: Das Data Transfer and Transformation Pattern

anhand der Definitionen aus dem Kapitel 5.1 zu beschreiben definieren wir folgende Mengen:

$Q'=\{q'_1, ..., q'_{s'}\} \subseteq Q$ mit $j' \in \{1, ..., s'\}$, $s' \geq 1$ und $s' \leq s$ als eine Teilmenge von Q

$P'=\{p'_1, ..., p'_{r'}\} \subseteq P$ mit $k' \in \{1, ..., r'\}$, $r' \geq 1$ und $r' \leq r$ als eine Teilmenge von P

Beim DTTP betrachten wir zunächst die Datenquellen q'_1 bis $q'_{s'}$ und die Datensenken p'_1 bis $p'_{r'}$. Dann betrachten wir in jeder entsprechenden Datenquelle bzw. Datensenke jeweils eine Teilmenge ihrer Datencontainer. Für die erste betrachtete Datenquelle q'_1 ist es z.B eine Teilmenge aus der Menge ihre Datencontainer $\{A_{1_1} \ ... \ A_{1_{y_1}}\}$ und für die letzte Datenquelle $q'_{s'}$ eine Teilmenge aus der Datencontainermenge $\{A_{s'_1} \ ... \ A_{s'_{y_s}}\}$. Für die

Datensenken ist es z.B eine Teilmenge aus der Datencontainermenge $\{B_{1_1}$ ... $B_{1_{m_1}}\}$ der ersten Datensenke p'_1 und eine Teilmenge aus der Datencontainermenge $\{B_{m_1}$ ... $B_{r'_{mr}}\}$ der letzten betrachteten Datensenke $p'_{r'}$. Beim DTTP übertragen und transformieren wir Daten *von einem oder mehreren Datencontainern einer oder mehrerer Datenquellen auf einen oder mehreren Datencontainer einer oder mehrerer Datensenken. Anschließend werden diese Daten in den Containern der Datensenken {p'_1 bis $p'_{s'}$} abgespeichert*. Dies geschieht mittels eines ETL-Prozesses, der wiederum aus einem Ablauf von ETL-Operationen besteht. Es werden die ETL-Operationen *Selektion, Projektion, Datenkonvertierung, Laden, Aggregation, Join, Union, Merge, Einfache Anreicherung und Komplexe Anreicherung* (siehe Kapitel 5.2) eingesetzt.

Hier ist es zu beachten, dass die Schnittmenge von Q' und P' nicht leer sein muss. Es kann also vorkommen, dass es Datenressourcen gibt, die sowohl als Datenquellen als auch als Datensenken auftreten können (z.B eine Datenbank). Im konkreten Fall des DTTP's heißt das, dass man z.B auch Daten aus einer Datenressource extrahieren und wieder in dieser Datenressource speichern kann.

5.3.1 Container-to-Container Pattern

Das *Container-to-Container Pattern (C2CP)* stellt die erste Unterklasse des DTTPs dar. Abbildung 5.3 zeigt das C2CP. Bei dem C2CP betrachten wir zunächst die Datenquelle q'_1 und einen in dieser Datenquelle verwalteten Datencontainer A_{1_1}. Danach betrachten wir die Datensenke p'_1 und einen Datencontainer B_{1_1} in dieser Datensenke. Beim C2CP übertragen und transformieren wir Daten von *einem Datencontainern einer Datenquelle auf einen Datencontainer einer Datensenke. Anschließend werden die Daten im Datencontainer der Datensenke p'_1 abgespeichert* [RSM]. Dies erfolgt mittels eines ETL-Prozesses, der wiederum aus einem Ablauf von ETL-Operationen besteht. Hierbei verwenden wir Operationen, die nur eine Eingabemenge bekommen (unäre Operationen). Das sind die ETL Patterns/Operationen *Selektion, Projektion, Datenformatkonvertierung , Laden, Aggregation und einfache Anreicherung* (siehe Kapitel 5.2).

Die Selektion der Geometrie-Daten aus der CSV-Datei, die Konvertierung dieser in einem für die Durchführung der Pandas-Simulation (siehe Kapitel 4.2.) geeignetes Datenformat und das Laden der konvertierten Daten in einer anderen Datei in der Pandas-Umgebung stellt ein Anwendungsbeispiel dieses Patterns und damit auch des DTTPs dar.

5.3.2 Data Split Pattern

Das *Data Split Pattern (DSP)* bildet die zweite Unterklasse des DTTPs . Auf Abbildung 5.4 wird das DSP dargestellt.

Bei dem DSP betrachten wir zunächst die Datenquelle q'_1 und einen darin verwalteten Datencontainer A_{1_1}. In diesem Datencontainer ist die Datenmenge S_{1_1} gespeichert. Danach betrachten wir eine Teilmenge der Datencontainermenge der Datensenken $\{p'_1 ... p'_{r'}\}$.

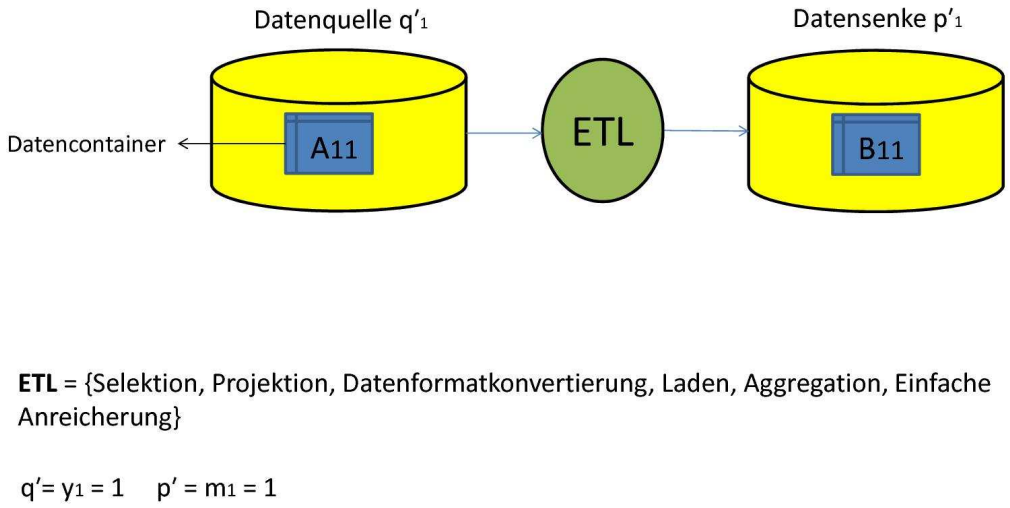


Abbildung 5.3: Das Container-to-Container Pattern

Beim DSP wird zunächst die Datenmenge S_{1_1} in u (typischerweise disjunkte) Teilmengen $T_l \subseteq S$, mit $l \in \{1, .., u\}$ und $u > 1$ aufgespaltet (selektiert). Anschließend übertragen, transformieren und verteilen wir die Teilmengen T_l auf *auf mehrere Datencontainer einer oder mehrerer Datensenken* [RSM], wobei jede Teilemenge T_l in genau einem Datencontainer gespeichert wird. Dies geschieht mittels eines ETL-Prozesses, der wiederum aus einem Ablauf von ETL-Operationen besteht. Hierbei kommen die ETL Patterns/Operationen *Selektion, Projektion, Datenkonvertierung, Laden und einfache Anreicherung* (siehe Kapitel 5.2) zum Einsatz.

Dieses Pattern findet seine Anwendung, wenn man nach Anwendung des DSPs komplexe Berechnungen durchführen muss. Ein Beispiel solcher komplexen Berechnungen stellt die parallele Verarbeitung der Teilmengen $T_l \subseteq S_{1_1}$ dar, indem man diese auf mehreren Datensenken verteilt. Die Verteilung der einzelnen Datenmengen T_l auf die Datencontainer der Datensenken hängt vom Parallelisierungsgrad ab. Will man *eine vollständige* Parallelisierung anstreben, dann verwendet man bei jeder beteiligten Datensenken genau einen Datencontainer, so dass jede Teilmenge T_l auf einer anderen Datensenken gespeichert wird und es gilt $k' = u$. Das bedeutet, dass die Anzahl der zu verteilenden Teilmengen T_l gleich der Anzahl der Datensenken ist, auf deren Datencontainern die Teilmengen T_l verteilt

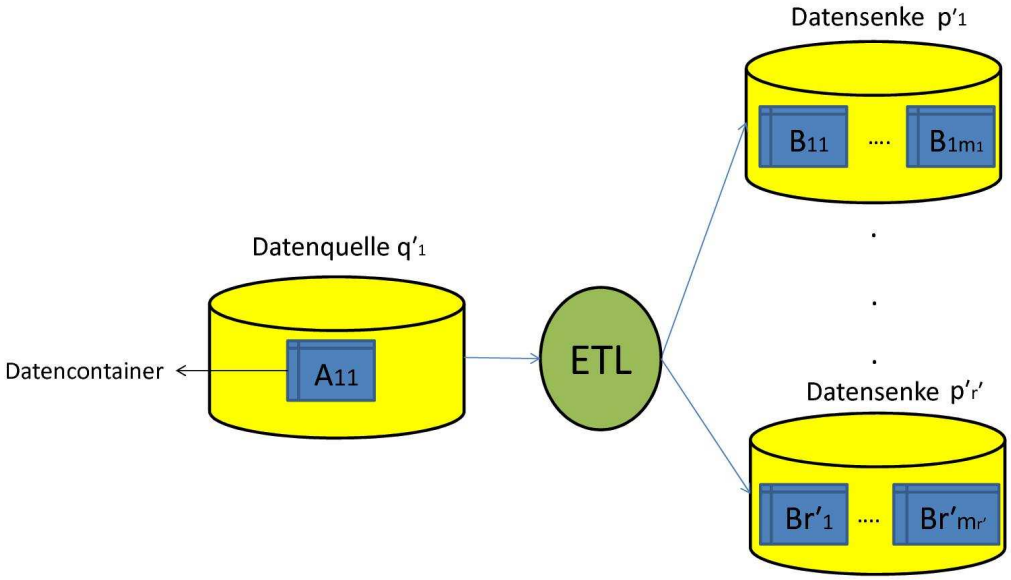


Abbildung 5.4: Das Data Split Pattern

werden. Wird eine *teilweise Parallelisierung* angestrebt dann verendet man bei zwei oder mehreren Datensenken mehrere Datencontainer, um die Teilmengen T_l abzuspeichern und es gilt 1<k'<u. Das bedeutet, dass die Anzahl der Datensenken, auf deren Datencontainern die Teilmengen T_l verteilt werden, kleiner der Anzahl der zu verteilenden Teilmengen T_l ist. Wird *keine Parallelisierung* angestrebt dann werden alle Teilmengen T_l auf Datencontainer genau einer Datensenke abgespeichert und es gilt k'=1. Im allgemeinen Fall gilt $k' \leq u$.

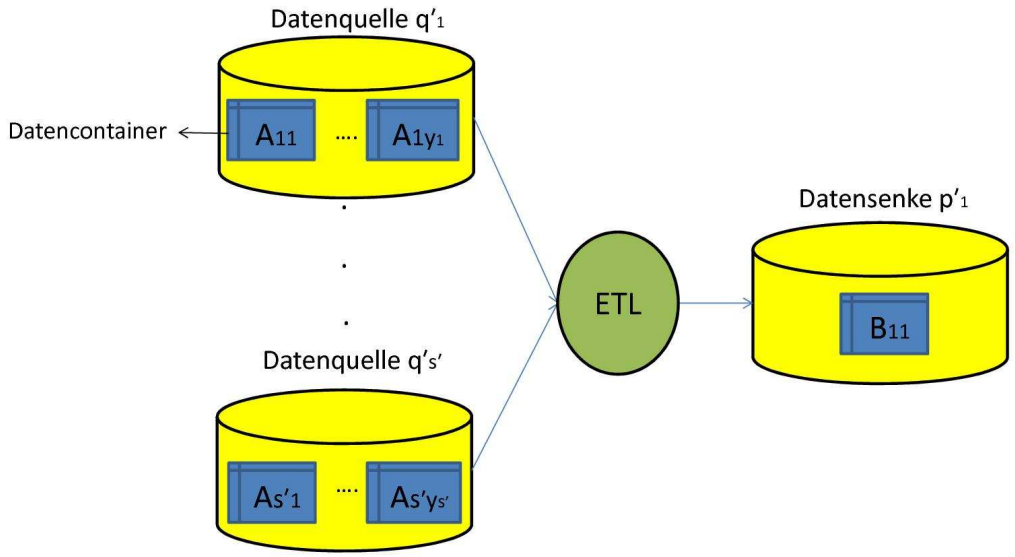
Bei temporären Datencontainern (siehe Kapitel 5.1) muss zuerst die Datenmenge S über die Durchführung eines Data Merge Pattern aus mehreren Datencontainern gebildet und anschließend in einem temporären Datencontainer gespeichert werden, bevor man das DSP anwenden kann.

Einen Anwendungsfall dieses Patterns und damit auch des DTTPs stellt die Durchführung von Simulationen auf einer multiskalaren Ebene dar. Bei dem Anwendungsszenario Pandas-Matlab Kopplung (siehe Kapitel 4.3) entspricht die Datenmenge S dem Simulationsraum auf der Pandas-Ebene. Die Elemente der Datenmenge S sind Gitter-Elemente, die auf der Ebene der Pandas-Simulation berechnet wurden. Die Menge der Gitter-Elementen wird nun in mehreren Teilmengen aufgespaltet, wobei eine einzelne Teilmenge (ein

Gitter-Element) eine bestimmte Anzahl von Gauss-Punkten enthält. Anschließend werden die einzelne Teilmengen aus der Pandas-Umgebung auf die Eingabedateien der einzelnen Matlab-Instanzen übertragen, transformiert und in diesen abgespeichert. Nachdem die Eingabedaten zur Verfügung gestellt wurden, können die einzelnen Teilmengen von den einzelnen Matlab-Instanzen parallel berechnet werden.

5.3.3 Data Merge Pattern

Das *Data Merge Pattern (DMP)* ist das Gegenstück des DSPs und bildet die letzuete Unterklasse des DTTPs. Auf Abbildung 5.5 wird das DMP dargestellt .



ETL = {Selektion, Projektion, Datenformatkonvertierung , Laden , Aggregation, Join , Union , Merge, einfache und komplexe Anreicherung}

Abbildung 5.5: Das Data Merge Pattern

Bei dem DMP betrachten wir zunächst die Datenquellen $\{q'_1 ... q'_{s'}\}$ und jeweils eine Teilmenge der in diesen Datenquellen verwalteten Datencontainern. In diesen Datencontainern sind Teilmengen $T_l \subseteq S$ mit $l \in \{1, .., u\}$ und $u > 1$ gespeichert. Danach betrachten wir die Datensenke p'_1 und den Datencontainer B_{1_1} in dieser Datenquelle. Beim DMP werden zunächst die Teilmengen T_l, die *in einem oder mehreren Datencontainer mehrerer Datenquellen* gespeichert sind, in eine Datenmenge S gemerget. Diese Datenmenge S wird dann auf den *Datencontainer* B_{1_1} *der Datensenke* p'_1 übertragen, transformiert und anschließend in diesem

abgespeichert [RSM]. Dies geschieht mittels eines ETL-Prozesses, der wiederum aus einem Ablauf von ETL-Operationen besteht. Hierbei kommen die ETL Patterns/Operationen *Selektion, Projektion, Datenkonvertierung, Laden, Join, Union, Merge, Aggregation, einfache Anreicherung und komplexe Anreicherung* (siehe Kapitel 5.2) zum Einsatz.

Wie oben erwähnt bildet das DMP das Gegenstück des DSPs. Hier werden die Ergebnisse der parallelisierten Berechnungen aus den Datencontainern der beteiligten Datensenken extrahiert und anschließend in einer Datenmenge S zusammengepackt. Das Extrahieren der einzelnen Ergebnisdaten aus den einzelnen Datencontainern der Datenquellen hängt vom Parallelisierungsgrad ab. Wurde bei der Verteilung der Teilmengen T_l auf die Datencontainer eine *vollständige Parallelisierung* angestrebt, dann wird jeweils eine Datenmenge T_l aus einem Datencontainern extrahiert und es gilt $j' = u$. Das bedeutet, dass die Anzahl der zu extrahierenden Teilmengen T_l gleich der Anzahl der Datenquellen ist, auf deren Datencontainern die Teilmengen T_l gespeichert sind. Wurde eine *teilweise Parallelisierung* angestrebt, dann werden mehrere Teilmengen T_l aus mehreren Datencontainern zweier oder mehrerer Datenquellen extrahiert. In diesem Fall gilt es 1<j'<u . Das bedeutet, dass die Anzahl der Datenquellen, auf deren Datencontainern die Teilmengen T_l gespeichert sind, kleiner der Anzahl der zu extrahierenden Teilmengen T_l ist. Wurde *keine Parallelisierung* angestrebt, dann werden alle Teilmengenangen T_l aus Datencontainern genau einer Datenquelle extrahiert und es gilt j'=1. Im allgemeinen Fall gilt $j' \leq u$.

Bei temporären Datencontainern (siehe Kapitel 5.1) muss zuerst die Teilmengen *Tl* über die Durchführung eines Data Splitt Pattern auf einem oder mehreren temporären Datencontainer der beteiligten Datensenken verteilt und abgespeichert werden, bevor man das DMP anwenden kann.

Ein Anwendungsbeispiel dieses Patterns findet man bei dem Anwendungsszenario der Pandas-Matlab Kopplung (siehe Kapitel 4.3). Die Teilmengen T_l entsprechen den verteilten Gauss-Punkten auf den einzelnen Matlab-Instanzen, wobei eine einzelne Teilmenge T_l eine bestimmte Anzahl von Gauss-Punkten enthält. Die auf jeder Matlab-Instanz verteilten Gauss-Punkte werden nun von den einzelnen Matlab-Instanzen berechnet. Die Ergebnisse werden anschließend in jeder Matlab-Instanz auf zwei Ausgabedateien kopiert. Von einer davon (Beschreibung der Gauss-Punkte) wird dann der Inhalt dieser Datei mit denen der anderen Dateien der anderen Matlab-Instanzen gemerget. Die sich daraus ergebene Datenmenge (Datenmenge S), entspricht dem Ergebnis bestimmter Matlab-Simulationszeitschritte. Das Ergebnis wird nun in eine Ausgabedatei kopiert und von dort aus in eine Tabelle in der Datenbank geladen.

5.4 Data Iteration Pattern

Das *Data Iteration Pattern (DIP)* bildet die zweite Art der im Rahmen dieser Arbeit vorgestellten Patterns. Auf Abbildung 5.6 wird das DIP dargestellt:

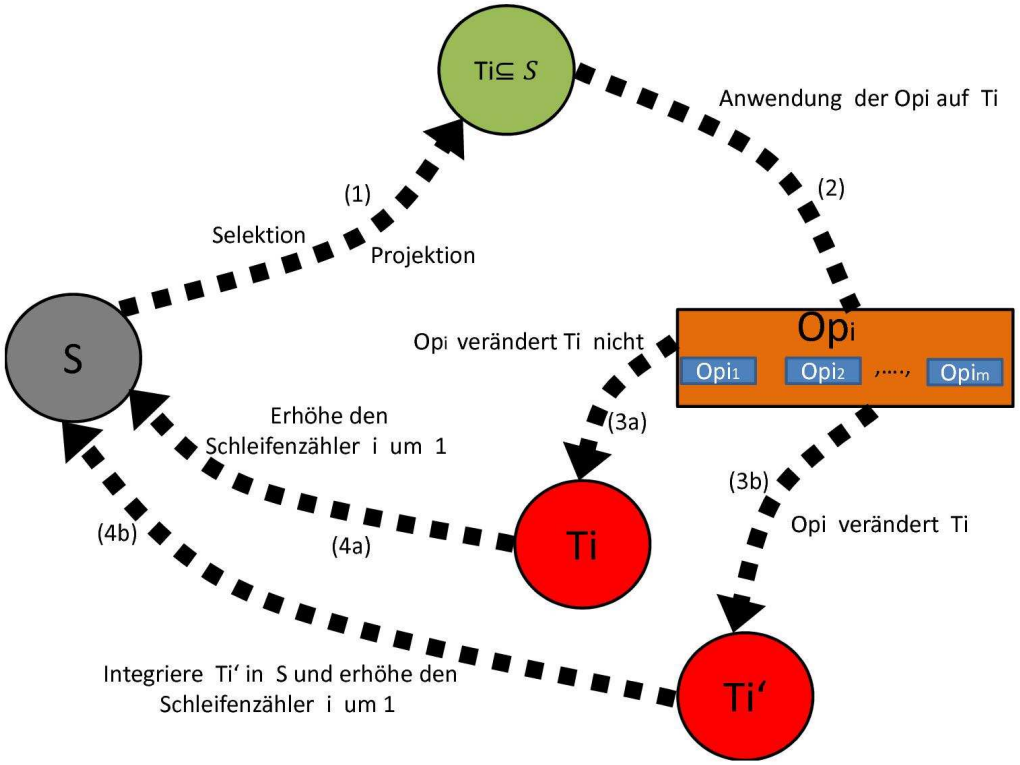


Abbildung 5.6: Das Data Iteration Pattern

Gegeben sei eine Datenmenge $S = \{s_1, ..., s_w\}$ mit w identifizierbaren Datenelementen s_1 bis s_w. Wie im Kapitel 5.1 beschrieben ist diese Datenmenge die Teilmenge eines kartesischen Produkts von bestimmten Wertebereichen. Die einzelnen Elemente $s_1...s_w$ stellen Tupel $\in S$ dar. Die Wertebereiche können dabei einfache Wertebereiche (wie Integer) oder auch komplexe Datenstrukturen beschreiben. Bei komplexen Datenstrukturen gehen wir aber nicht tiefer in die Datenstrukturen hinein, da dies für das DIP nicht relevant ist.

Für jede Iteration $i \in \{1, .., n\}$ iteriert das DIP über diese Datenmenge S und nimmt über eine Selektions- bzw. Projektionsoperation (siehe Kapitel 5.2) eine Teilmenge $T_i \subseteq S = \{s'_{i_1}, s'_{i_2}, ..., s'_{i_{p_i}}\}$. Es gilt $p_i \leq w$, das heißt, dass bei jedem Iterationsschritt i die Anzahl der selektierten Elementen in T_i kleiner-gleich der Anzahl der Elementen in S ist. Anschließend wird eine Operation op_i auf T_i ausgeführt, wobei die Operation op_i aus mehreren Teiloperation $op_{i1}, op_{i2}, ..., op_{i_m}$ zusammengesetzt wird. Nach der Ausführung dieser Operation unterscheidet man zwei Fälle. Beim ersten Fall verändert die Operation op_i die Teilmenge T_i nicht. Hierbei kommt als Ergebnis z.B ein skalarer Wert (z.B ein Integer) raus und der Schleifenzähler wird nachfolgend um 1 erhöht, so dass die nächste Iteration startet. Beim zweiten Fall wird die Teilmenge T_i durch die Anwendung der Operation op_i verändert. Die Menge T'_i, die sich daraus ergibt, muss dann in S integriert werden. Darüber hinaus muss

der Schleifenzähler um 1 erhöht werden.

Es ist zu beachten, dass bei Veränderung der Teilmenge T_i über die Anwendung der Operation op_i und der Integration dieser in die Datenmenge S, die Datenmenge S_{i+1} auch verändert wird. Dies muss man bei der Selektion der Teilmenge T_{i+1} berücksichtigen, falls $T_i \cap T_{i+1} \neq \emptyset$.

Beim DIP betrachten wir Iterationen nur auf der Workflow-Ebene. Schleifen, die von Simulationsprogramen zur Durchführung irgendwelcher Berechnungen verwendet werden, werden als teil der Operation betrachtet. Diese Schleifen werden unter Umständen in einer übergeordneten Workflow-Schleife mehrmals iteriert. weiterhin gibt es mehrere Varianten, um die Anzahl der n Iterationen festzulegen. Manchmal wird die Zahl n in der Modellierungszeit festgelegt, indem der Modellierer einfach die Anzahl der Iterationen bestimmt. Manchmal wird sie erst zur Laufzeit festgelegt, indem man z.B ein regulärer Ausdruck, z.B ein XPath-Ausdruck angibt, der zum Startzeitpunkt der Schleife die Zahl n berechnet. Manchmal steht die Zahl n aber erst am Ende der Schleife fest. Das ist z.B. der Fall, wenn man eine Abbruchbedienung in einer While-Schleife oder RepeatUntil-Schleife festlegt, z.B wenn die Datenqualität eine bestimmten Schwelle erreicht/überschritten hat.

Ein Anwendungsbeispiel dieses Patterns stellt die Lösungsphase der Pandas-Simulation (siehe Kapitel 4.2) dar, wenn Pandas in seine Simulationsschleife gerät. Zur Vereinfachung der Beschreibung des Anwendungsfalls, wird nur auf die Berechnung der FEM-Gitter und der Ergebnisse während des Durchlaufs eines Workflow-Iterationsschrittes i eingegangen. Weitere Berechnungen, die Pandas durchführt, wie z.B die Berechnung der Matrizen oder der Vektoren, werden nicht berücksichtigt. Für andere Beispiele müssen unter Umständen auch andere Daten berücksichtigt werden. Dafür muss die Beschreibung des DIPs entsprechend angepasst werden. Die Datenmenge S besteht aus folgenden Mengen:

1. Zeitintervalle= $\{Z_1, Z_2, ..., Z_n\}$
2. FEM-Gitter= $\{Gitter_0, Gitter_1 ... Gitter_{p_1} ... Gitter_{p2} ... Gitter_{p_n}\}$
3. Ergebnisse= $\{E_0, E_1 ... E_{p_1} ... E_{p_2}, ..., E_{p_n}\}$

Der Workflow iteriert über die Menge der Zeitintervalle und extrahiert in jeder Iteration ein Zeitintervall. Die Menge FEM-Gitter entspricht der berechneten Gitter. Eine Iteration i entspricht der Ausführung eines Zeitintervalls. Innerhalb eines Zeitintervalls werden mehreren Simulationszeitschritte durchgeführt, wobei in jedem Zeitschritt ein Gitter und ein Simulationsergebnis berechnet wird. In der Tabelle 5.1 werden die Zeitintervalle den Simulationszeitschritten zugeordnet. Nach einem bestimmten Iterationsschritt i sieht die Datenmenge S wie foglt aus:

$S_i = \{Zeitintervalle, Gitter_i, Ergebnisse_i\}$ mit $i \in \{1...n\}$.

Der Zeitschritt 0 im ersten Zeitintervall entspricht der Initialisierung der Simulation. Zu diesem Zeitpunkt sieht die Menge S wie folgt aus:

Zeitintervalle	Simulationszeitschritte
1	0...$[p_1]$
2	$(p_1...p_2]$
...	...
i-1	$(p_{i-2}...p_{i-1}]$
i	$(p_{i-1}...p_i]$
...	...
n-1	$(p_{n-2}...p_{n-1}]$
n	$(p_{n-1}...p_n]$

Tabelle 5.1: Zuordnung der Simulationszeitschritten in Zeitintervallen

$S_0 = \{Zeitintervalle_1, Gitter_0, Ergebnis_0\}$

Das $Gitter_0$ wird in diesem Fall mit dem Wert 0 belegt. Daher ist das $Ergebnis_0$ ebenfalls 0. Zu jedem Zeitschritt $j \in \{0...p_m\}$ wird ein Gitter erzeugt oder angepasst, so dass die Basis zur Aufstellung der Matrix-Gleichung $A_j \cdot x_j = b_j$ gebildet wird. Zur Generierung des Endergebnisses wird die Gleichung anschließend gelöst. Für die Berechnung der Gitter innerhalb eines Zeitintervalls i $\in$ {1...n} stellt die Menge S_{i-1} die Menge S_i vor dem Zeitintervall i dar. In dieser Menge sind alle bis dahin berechneten Gitter und Ergebnisse enthalten. Die Menge S_{i-1} sieht wie folgt aus:

S_{i-1} = {{Zeitintervalle}, {$Gitter_0$, $Gitter_1$,..., $Gitter_{p_{i-1}}$}, {$Ergebnis_0$, $Ergebnis_1$,..., $Ergebnis_{p_{i-1}}$}}

Die Menge S_i entspricht der Menge nach dem i-ten Zeitintervall, deren Zeitschritte berechnet werden sollen. Wir betrachten die Berechnungen aller einzelnen Zeitschritte innerhalb des Zeitinetrvalls i als Black-Box und wir zeigen wie die Menge S_i nach der Berechnung aller Zeitschritten aussieht. Man muss zunächst die Teilmenge T_i bilden, indem man eine Projektion und eine Selektion zusammen auf die Datenmenge S_{i-1} anwendet. Damit hat man die Teilmenge T_i = {{Zeitschritt$_i$}, {$Gitter_0$, $Gitter_1$,..., $Gitter_{p_{i-1}}$} , $\emptyset$} gebildet. Auf diese Menge wird nun die Operation op_i angewendet. Nach der Berechnung aller Zeitschritten innerhalb des Intervalles i sieht die Menge T_{i-1} wie folgt aus:

T_i' ={{$Zeitintervalle_i$}, {$Gitter_0$, $Gitter_1$,..., $Gitter_{p_{i-1}}$,..., $Gitter_{p_i}$}, {$Ergebnis_0$, $Ergebnis_1$,..., $Ergebnisse_{p_{i-1}}$,..., $Ergebnisse_{p_i}$}}

In diesem Fall verändert sich die Teilmenge T_i. Aus der Anwendung der op_i auf die ursprüngliche Teilmenge T_i={{$Zeitschritti$},{$Gitter_{p_i}$} hat sich die neue Menge T_i' ergeben. Also die Elemente $Gitter_{p_i}$ $Ergebnis_0$, $Ergebnis_1$,..., $Ergebnisse_{p_{i-1}}$,..., $Ergebnisse_{p_i}$, ist das neue Element, das der ursprünglichen Menge T_i über die Anwendung der Operation op_i hinzugefügt wurden. Anschließend wird die neue Datenmenge T_i' in der Datenmengen

S_i integriert. Diese sieht wie folgt aus:

S_i ={{$Zeitintervalle_i$}, {$Gitter_0$, $Gitter_1$,..., $Gitter_{p_{i-1}}$,..., $Gitter_{p_i}$}, {$Ergebnis_0$, $Ergebnis_1$,..., $Ergebnisse_{p_{i-1}}$,..., $Ergebnisse_{p_i}$}}

Die Menge S wurde also um alle berechneten Gitter und Ergebnisse erweitert. Es wird bei den nächsten Iterationen der gleiche Prozess wiederholt, bis der letzte Iterationschritt n-1 durchgeführt wurde und alle Simulationszeitschritte 0 bis p_n berechnet wurden.

Im Rahmen dieses Kapitels haben wir die aus den Anwendungsszenarien identifizierten Datenmanagementpatterns formalisiert und erläutert. Diese Datenmanagement Patterns müssen auf ausführbare Workflow-Fragmente abgebildet werden. Diese Thematik wird der Gegenstand des nächsten Kapitels sein.

6 Transformation der Datenmanagementpatterns auf ausführbare Workflow-Fragmente

Im folgenden Kapitel wird auf die Transformation der DM-Patterns auf ausführbare Workflow-Fragmente eingegangen. Im Teilkapitel 6.1 werden wir zunächst den in dieser Arbeit verwendeten Mechanismus für die Abbildung der DM-Patterns anhand einer fünf Schichten-Architektur und des Pattern-Transformers erläutern [RM11] [Rei11]. Im Teilkapitel 6.2 werden Beispiele mittels der Anwendungsszenarien aus Kapitel 4 und der im Kapitel 5 definierten DM-Patterns vorgestellt. Im Teilkapitel 6.3 werden wir die definierte Kontrollstrategie zur Abbildung der Patterns auf die Workflow-Fragmente erklären. Zum Schluss werden wir über die Architektur des Abbildungsmechanismus diskutieren.

6.1 Abbildungsmechanismus und Pattern-Transformer

Im Kapitel 5.1 haben wir eine vorläufige Architektur zur Pattern-Hierarchie vorgestellt und über die bis dahin bekannten Schichten diskutiert. Nachdem wir die neuen DMPs im Kapitel 5 formalisiert und erläutert haben, können wir jetzt diese Hierarchie vervollständigen, indem wir die neue DMPs in der Patternhierachie hinzunehmen (siehe Abildung 6.1). Auf der dritten Schicht befinden sich die *Basis Datamanagementpatterns (BDPs)*, nämlich das DTTP und seine Unterklassen sowie das DIP. Die BDPs entsprechen der minimalen Menge von DMPs, die für Simulationsworkflows relevant sind, und bilden die Grundlage zur Definition anderer Mengen. Auf der zweitobersten Schicht befinden sich die *Zusammengesetzten Datenmanagmentpatterns (ZDPs)*. Diese Abstraktionsebene haben wir geschaffen, um die Zusammenfassung mehrerer BDPs in einem anderen abstrakteren DMP zu ermöglichen. Auf diese Art und Weise können beispielsweise fünf hintereinander auszuführende C2CPs in nur einem Schritt auf ein ausführbares Workflow-Fragment abgebildet werden. Weiterhin erleichtert dies das Leben des Modellierers, da er nicht wiederholt eine Reihe von Datenmanagementoperationen gegen bestimmten Datenressourcen modellieren muss und da er die abstrakteren ZDPs i.d.R auch auf abstraktere Weise und damit einfacher modellieren kann. Wie es sich aus der Abbildung 6.1 erkennen lässt, gibt es mehrere Möglichkeiten, um einen bestimmtes Datenmanagementpattern auf ein ausführbares Workflow-Fragment abzubilden. Diese mehrstufige Abbildung der DMPs basiert auf Anwendung von Regeln [Rei11]. Hierbei gibt es zwei Typen von Transformationsregeln. Bei dem *Regeltyp-1* wird ein Pattern direkt auf ein ausführbares Workflow-Fragmente transformiert. Diese Transformation wird entweder über DM-Aktivitäten im SIMPL-Rahmenwerk (siehe Kapitel 3.2) oder Service-Aufrufe

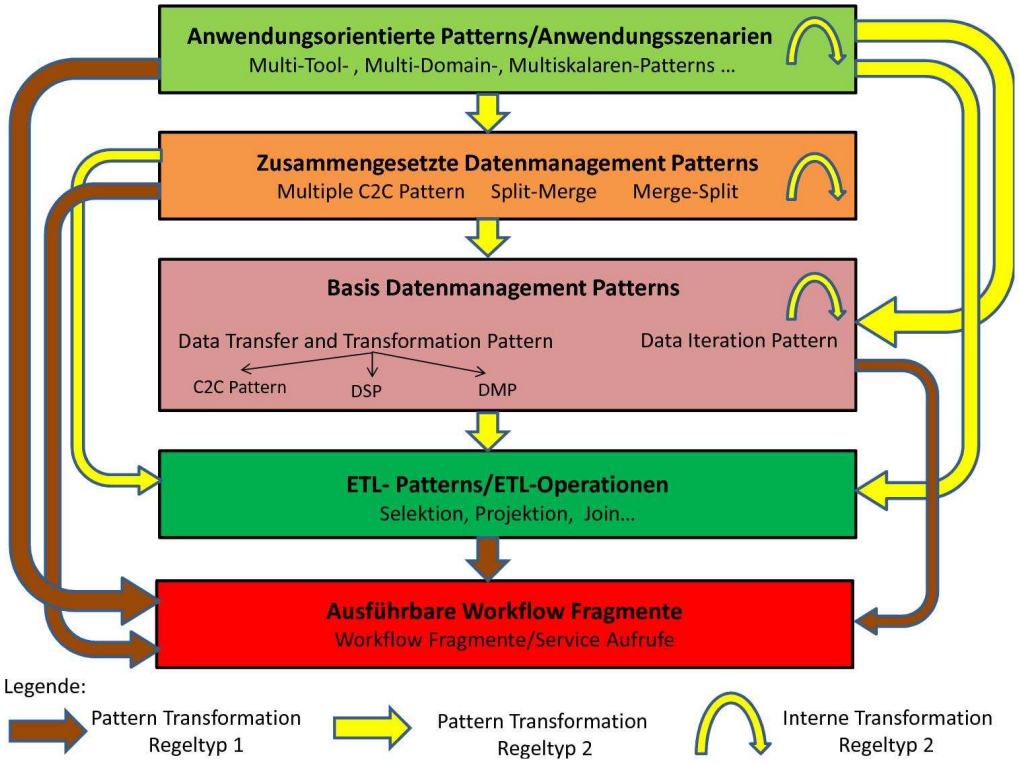


Abbildung 6.1: Das Konzept zur Transformation der Patterns auf ausführbare Workflow-Fragmente vgl.[RM11]

realisiert. Das bedeutet, dass die Workflow-Fragmente entweder existieren oder sie müssen zuerst über die Unterstützung von Metadaten generiert werden. So kann beispielsweise von der Schicht der *ETL-Patterns/Operationen* aus eine direkte Transformation möglich sein, denn es geht hier um *feingranulare Patterns*, für die bereits Workflow-Fragmente existieren oder generiert werden können.

Bei dem *Regeltyp-2* wird ein Pattern auf ein Workflow-Fragment mit anderen, meist feingranulare Patterns abgebildet. In diesem Fall muss man die Patterns u.U über ein iteratives Verfahren verfeinern, um sie in eine solche Form umzuwandeln, so dass eine direkte Transformation über einen Regeltyp-1-Ansatz realisierbar ist. Diese Transformation kann entweder zwischen mehreren Schichten und/oder innerhalb einer einzelnen Schicht stattfinden. Man kann beispielsweise von der Schicht der BDPs aus ein C2CP in ein oder mehrere ETL Patterns iterativ verfeinern. Auf dieser Art und Weise kann einen Regeltyp-1-Ansatz von Schicht der ETL-Patterns/Operationen aus realisierbar sein.

Bis jetzt haben wir das Konzept der Pattern-Transformation abstrakt beschrieben. Jetzt wollen wir auf diese Thematik detaillierter eingehen und erklären, welcher Mechanis-

mus angewendet wird, um die Abbildung der Patterns mittels eines *Pattern-Transformers* zu realisieren. Diese Beschreibung basiert auf [Rei11].

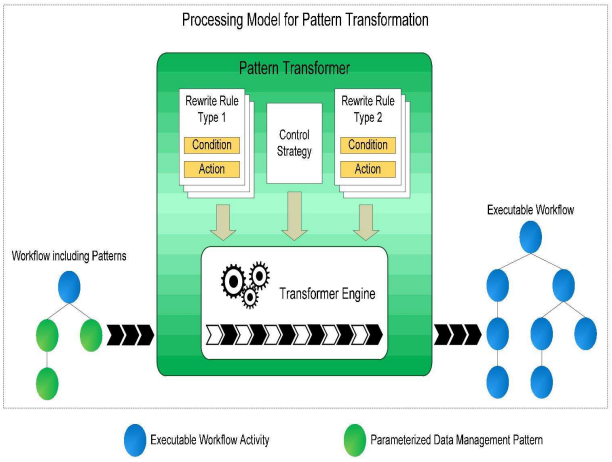


Abbildung 6.2: Die Pattern-Transformation mittels des Pattern-Transformers [Rei11]

Auf Abbildung 6.2 ist der Pattern-Transformer und die Transformer Engine-Komponente zu sehen. Links befindet sich das zu transformierende Workflow mit Patterns. Mit der Farbe blau ist eine ausführbare Workflow-Aktivität gekennzeichnet (z.B eine BPEL-Receive-Aktivität) und mit grün ein parametrisiertes, nicht ausführbares DMP (z.B ein C2CP). Dieser Workflow soll nun mit Hilfe der *Transformer Engine* auf einen ausführbaren Workflow (rechts auf dem Bild) transformiert werden. Hierbei traversiert man durch diesen Workflow-Graphen und versucht für jedes Pattern eine passende Transformationsregel zu finden. Jede Transformationsregel besitzt einen *Condition Part* und einen *Action Part*. Im Condition Part werden die Bedingungen beschrieben, unter denen eine gewisse Transformationsregel auf ein DMP angewendet werden darf. Diese Bedingungen hängen zum einen mit den Parameterwerten des zu transformierenden Patterns und zum anderen mit den Metadaten, die die beteiligten Datenressourcen und Workflow-Fragmente beschreiben, zusammen. Über die Parameter werden z.B. die Datenressourcen und die Datencontainer festgelegt, auf die zur Ausführung einer bestimmten Datenmanagmentoperation zugegriffen werden soll, wobei das Pattern selbst diese Datenmangementoperation definiert. Die Metadaten beschreiben z.B funktionelle oder nicht-funktionelle Eigenschaften der Datenressourcen.

Im Action Part einer Regel wird festgelegt, wie dieses DMP auf ein ausführbares Workflow-Fragment abgebildet wird, wenn die Regel auf das DMP angewendet wird. Wie bereits in diesem Kapitel erwähnt werden zur Transformation der DMPs zwei Typen von Regeln festgelegt: Der Regeltyp-1 und der Regel-Typ-2. Zur Beschreibung und Funktionsweise der Anwendung der Regeln ist der Leser auf die Erläuterung der Pattern-Hierarchie am Anfang dieses Kapitels verwiesen.

Über die *Kontrollstrategie* wird gesteuert, in welcher Reihenfolge die Transformati-

onsregeln auf Anwendbarkeit überprüft werden. Zuallererst muss die Menge der Regeln, die auf ein gewisses DMP überhaupt anwendbar sind, festgestellt werden. Man könnte beispielsweise andere Regeln für die Transformation eines ETL-Patterns als bei der Transformation eines C2CPs anwenden. Im ersten Fall wird z.B eine Transformationsregel vom Typ1 festgelegt. Im zweiten Fall kann aber die Anwendung einer Regel von Typ-2 notwendig sein. Darüber hinaus muss man die Reihenfolge der Anwendung der prinzipiell anwendbaren Regel auf dieses DMP bestimmen. Im Kapitel 6.3 werden wir auf diese Thematik detaillierter eingehen.

Unter Berücksichtigung der Pattern-Hierarchie kann man zur Realisierung dieses Konzepts zwei verschiedene Ansätze verfolgen. Man kann die Hierarchie entweder von unten nach oben oder von oben nach unten durchlaufen. Bei einem *Bottom-Up-Ansatz* versucht man, ausgehend von der Schicht der Workflow-Fragmente, die DMPs auf den einzelnen Schichten und deren Beziehungen, sowohl innerhalb einer Schicht als auch zwischen den einzelnen Schichten, möglichst abstrakt zu beschreiben. Dabei geht es um eine *Abstraktionsunterstützung für den Modellierer*, damit man ihm möglichst viel Arbeit abnimmt. Bei dieser Abstraktionsunterstützung verliert man Informationen über die involvierten DM-Operationen. Das könnte aber zur Folge haben, dass man bei dem Top-Down Ansatz nicht mehr genügend Informationen über die einzelnen Schichten und deren Beziehungen hat, um die DMPs auf die ausführbaren Workflow-Fragmente abzubilden.

Bei einem *Top-Down Ansatz* geht man von der ersten Schicht der DMPs aus. Dabei geht es um die Transformation der DMPs auf die ausführbaren Workflow-Fragmente. Anhand der Informationen auf der jeweils höheren Ebene müssen die geeigneten DM-Operationen für die nächste Ebene definiert werden, um die Patterns auf die Workflow-Fragmente abzubilden. Das kann aber zur Folge haben, dass die Abstraktionsunterstützung für den Modellierer eingeschränkt wird. Außerdem muss man möglicherweise die auf einer höheren Schicht fehlenden Informationen beim Top-Down-Ansatz ergänzen.

Um den obigen Sachverhalt zu verdeutlichen, werden wir ein Beispiel anhand der Ebene der ETL-Patterns angeben. Bei einem Top-Down Ansatz versucht man die Datenmanagementoperation möglichst detailliert zu beschreiben, um die ETL-Patterns auf die Workflow-Fragmente transformieren zu können. Wenn man aber einem Bottom-Up Ansatz verfolgt, dann versucht man die ETL-Patterns möglichst abstrakt zu beschreiben, damit der Modellierer nicht viele Details über die Datenmanagementoperationen definieren muss. Deshalb gibt es einen Zielkonflikt zwischen der Abstraktionsunterstützung für den Modellierer und der Möglichkeit alles ausführbar zu halten. In diesem Zusammenhang stellen sich folgende Fragen:

1. Wenn man dem Bottom-Up Ansatz verfolgt, welche Informationen auf den unteren Ebenen kann man weg lassen, um die Modellierung der Patterns auf einer höheren Ebenen möglichst abstrakt zu halten?
2. Wenn man dem Top-Down Ansatz verfolgt, wie kommt man bei den unteren Ebenen an die nötigen Informationen, die auf einer höheren Ebene noch fehlen?

In diesem Zusammenhang müssen voneinander abhängige Fragen beantwortet werden:

1. Wie sieht die Parametrisierung von bestimmten DMPs aus? Welche Parameter müssen vom Nutzer unbedingt festgelegt werden?
2. Wie siehen die Transformationsregeln, also der Condition und Action Part, aus? Wie wird die Kontrollstrategie definiert?
3. Wie sieht eine Architektur aus, die den Abbildungsmechanismus umsetzt? Welche Metadaten können bei der Parameterfestlegung und bei der Pattern-Transformation helfen?

Im nächsten Kapitel werden wir auf diese Fragen anhand von konkreten Beispielen eingehen.

6.2 Beispiele zur Pattern-Transformation

An dieser Stelle wollen wir anhand von Beispielen aus der Pandas pre- und postproseccing Phase zeigen, wie eine solche Pattern-Transformation, wie sie im Kapitel 6.1 beschrieben, aussieht.

6.2.1 Beispiele anhand der Pandas Preprocessing-Phase

Funktionsweise und Identifikation der Patterns

Bei diesem Beispiel verwenden wir den Bottom-Up Ansatz. Wir schauen erstmal die existierende Workflows an und wir identifizieren dabei die relevanten Workflow-Fragmente für die DMPs. Danach erarbeiten wir die Parametrisierung und Transformation der identifizierten DMPs.

In der *preprocessing-Phase* startet der Workflow mit dem Laden grundlegender Eingabe-Daten bezüglich des zu simulierenden Knochens aus verschieden Dateien (siehe Kapitel 4.2). Diese Dateien sind [rai]:

1. Die *Shape-Datei*, in der verschiedene mathematische Voraussetzungen für die Ausführung der FEM-basierten Simulation definiert sind.
2. Die *Geometry-Datei* für die Beschreibung der geometrischen Struktur des Knochens.
3. Die *ivars-Datei*, die verschiedene interne Informationen für die Visualisierung enthält.
4. Die *material Parameters-Datei*, in der die Parameter bezüglich des Materials des zu simulierenden Knochens gegeben sind.
5. Die *Command-Datei*, die eine Menge von Simulationsbefehlen enthält, die *Nodes-Datei*, in der die Koordinaten der Elemente im Gitter spezifiziert sind.

6. Die *Element-Datei* für die Spezifikation der Topologie der Elemente im Gitter.
7. Die *Nodes-Datei*, in der die Koordinaten der Elemente im Gitter spezifiziert sind.

Diese Dateien (Quell-Dateien) werden vom Arbeitsverzeichnis des Workflow-Rechners in das Arbeitsverzeichnis des Pandas-Rechners kopiert. In diesem Fall kann man eine Schleife definieren, die über eine *Liste von Dateipfaden* durchläuft. Jedes Dateipfad in dieser Liste entspricht dem Pfad zu einer Quell-Datei im Arbeitsverzeichnis des Workflow-Rechners. Für jeden Iterationsschritt wird ein Dateipfad aus der Liste ausgewählt. Anschließend werden die jeweiligen ausgewählte Dateien in den Arbeitsverzeichnis des Pandas-Rechners kopiert. Die Referenz auf das Arbeitsverzeichnis bleibt in jedem Iterationsschritt gleich. Die Schleife wird beendet, wenn man über alle Dateipfade der Liste durchgelaufen ist. Dieser Datentransfer kann man anhand der im Rahmen dieser Arbeit definierten Datenmanagementpatterns beschreiben. Abbildung 6.3 zeigt ein Workflow-Fragment, das die Pandas Preprocessing-Phase beschreibt. Dieses Workflow-Fragment stellt das Transformationsziel dar.

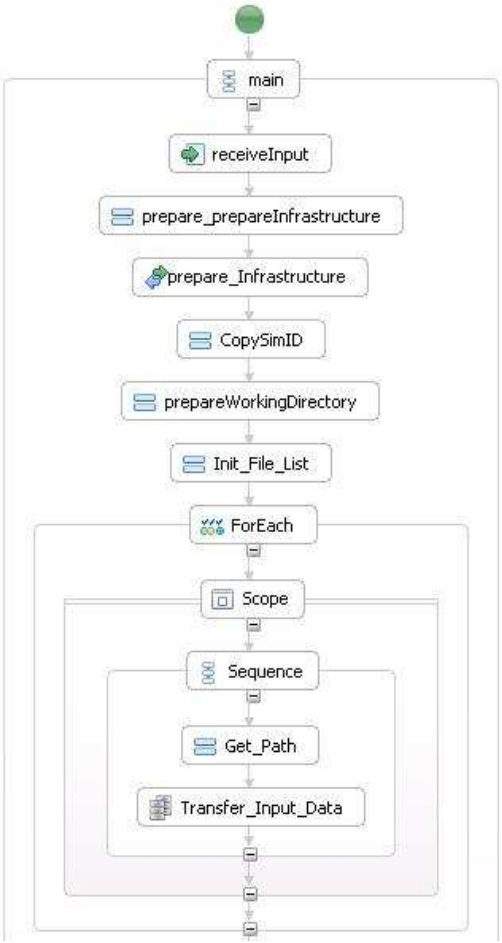


Abbildung 6.3: Workflow-Fragment zur Pandas-preprocessing-Phase

Über die ReceiveInput-Aktivität wird die SOAP-Nachricht (siehe Kapitel 2.1) empfangen . In der Assign-Aktivität *prepare_prepareInfrastructure* werden die Eingabedaten für den nachfolgenden Web-Service-Aufruf (siehe Kapitel 2.1.1) festgelegt. Über die *Invoke-Aktivität prepare_Infrastructure* wird ein Webservice (siehe Kapitel 2.1.1) aufgerufen, der die SmulationsID generiert und das Pandas-Arbeitsverzeichnis im Pandas-Rechner erzeugt [Dor11]. Dieser Web Service sendet diese Ergebnisse zurück an den Workflow. Die SimulationsId und der Pfad zum Arbeitsverzeichnis werden anschließend von der Message-Variable der prepare_Infrastructure-Aktivität auf die Variablen *SSimID*und *"workingDirectory"* in den Assign-Aktivitäten *CopySimID* und *prepareWorkingDirectory* kopiert. Anschließend wird die SimulationsID über eine XPath-Konkatenation [1] am Ende des Pfades zum Arbeitsverzeichnis hinzugefügt. Das Ergebnis der Konkatenation wird in der Variablen "workingDirectory"gespeichert. In der Assign-Aktivität *Init_File_List* wird die zuvor im BPEL-Dokument definierte Liste der Dateipfaden initialisiert. Die folgende Beschreibung der Liste wurde aus der Implementierung eines anderen Simulationsworkflows entnommen. Die Postprocessing-Phase läuft aber genau so wie bei der Pandas-Simulation ab.

```
<bpel:element="ns2:fileReferenceList"></bpel:variable>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.command</fileName>
  </fileReference>
  <fileReference
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.element</fileName>
  </fileReference>
  <fileReference>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.geometry</fileName>
  </fileReference>
  <fileReference>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.ivars</fileName>
  </fileReference>
  <fileReference>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.nodes</fileName>
  </fileReference>
  <fileReference>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.shape</fileName>
  </fileReference>
  <fileReference>
    <directory>C:\Users\hoosea\testdaten\</directory>
    <fileName>2d_wanne.material parameters</fileName>
  </fileReference>
```

Der Liste wird der Variablentyp *fileReferenceList* zugewiesen. Innerhalb der Liste werden alle Dateipfade abgelegt. Ein solches Dateipfad besteht aus dem (<directory>) und den

[1] http://www.w3.org/TR/xpath20/

Namen der jeweiligen Datei (<filename>). Die *foreach Aktivität* (siehe Kapitel 2.2) realisiert die Durchführung der Iteration. Im folgenden wird die forEach-Schleife anhand eines Ausschnittes aus dem Source-Code beschrieben.

```
<forEach counterName="Counter" name="ForEach">
   <startCounterValue>
     <![CDATA[1]]>
   <\startcounterValue>
<finalCounterValue ![CDATA[round(count($List/fileReference))]]> <\finalcounterValue>
<scope>
   <sequence>
     <assign name=\textit{"Get\_Path"}>
     <copy><from variable="Counter"></from>
      <to variable=\textit{"ListNumber"}></to>
<\copy>
<copy>
<from>
<concat($List/fileReference[number($ListNumber)]/directory/text(),
       $List/fileReference[number($ListNumber)]/fileName/text())>
</from>
<to variable="Input_Data_Path"></bpel:to>
</copy>
</assign>
<TransferData Activity name="Transfer_Input_Data" dsStatement="#Input_Data_Path#"
    dsKind="Windows Local" dsType="Filesystem" dsIdentifier=?"DataSourceLocal"
    dsLanguage="Shell" targetDsType="Filesystem" targetDsKind="SSH Server"
    targetDsIdentifier="DataSourceRemote" targetDsLanguage="Shell" targetDsContainer=
    "#workingDirectory#"></simpl:transferDataActivity>
</sequence>
</scope>
<\forEach>
```

Die forEach-Schleife soll von eins (Startwert) bis zur Anzahl der Dateipfade in der Liste (Endwert) iterieren. Hierbei wird in der BPEL-Variablen *"Counter"* der Index i des jeweils aktuellen Iterationsschrittes gespeichert. Der Startwert <*startCounterValue*> und der Endwert <*finalCounterValue*> werden über einen XPath-Ausdruck ausgewertet. Das Ergebnis dieser Auswertungen ist jeweils eine Integer Zahl. Diese Integer-Werte bleiben über den gesamten Schleifen-Durchlauf konstant. Für jeden Iterationsschritt i wird der Inhalt der Variable Counter um eins erhöht. Wenn der Wert von <*finalCounterValue*> erreicht wird, wird die Iteration beendet [OAS]. Ab <Scope> beginnt der Schleifenrumpf. Innerhalb der *Scope-Aktivität* werden alle Aktivitäten sequentiell nacheinander ausgeführt. Dies wird über die Aktivität *Sequence* realisiert [OAS]. Für jeden Iterationsschritt i wird ein Element (Dateipfad) aus der Liste selektiert. Dies erfolgt über die Assign-Aktivität *Get_Path*. Hierbei wird der i-te Iterationsschritt bzw. die Nummer des besuchten Pfades in der Liste von der Variablen *counter* auf die Variable *List Number* kopiert. Hierbei wird eine Datentyp-konvertierung von Int auf Integer, oder umgekehrt vorgenommen. Darüber hinaus wird über die XPath-konkatenation List/fileReference[number(ListNumber)]/directory/text(), List/fileReference[number(ListNumber)]/fileName/text()) der ganze Dateipfad aus der Liste ausgelesen und auf die Variable *Input_Data_Path* kopiert. Damit ist der selektierte Dateipfad bekannt. Die Ziel-Datei im Arbeitsverzeichnis des Pandas-Rechners ist auch

bekannt. Sie kann nämlich aus der Variable "#workingDirectory#"der Assign-Aktivität *prepareWorkingDirectory* (siehe oben) ausgelesen werden. Damit kann die BPEL-DM-Aktivität TransferData die Daten von der jeweiligen Quell-Datei im Eingabedatenverzeichnis in den Pandas-Arbeitsverzeichnis kopieren. Dieser Datentransfer erfolgt über die Angabe der Variablen *Input_Data_Path* und "#workingDirectory#"vom Nutzer. Weiterhin muss er die Datenressource, in der sich die Quell-Datei befindet, ebenfalls festlegen. Das geschieht über die Angabe der Variable *"DataResourceLocal"*, also diese Datenressource ist der lokale Rechner (Workflow-Rechner). Anschließend muss er die Datenressource, in der sich die Ziel Datei befindet festlegen. Das geschieht über die Angabe der Variable *"DataResourceRemote"*, also diese Datenressource ist der entfernte Pandas-Rechner. Der Typ der Datenressource (Filesystem), der Untertyp (Windows Local) und die Sprache (Shell), die das entsprechende Commando zum jeweiligen Kopieren der Daten ausführt, werden ebenfalls festgelegt. Diese Informationen werden allerdings nicht vom Nutzer angegeben, sondern sie werden automatisch über das Ressource Management (siehe Kapitel 3.1) geladen. Die forEach Schleife wird beendet, wenn alle Dateipfade aus der Liste selektiert wurden.

Nachdem wir das Workflow-Fragment zur preprocessing-Phase erläutert haben werden wir die DMPs aus diesem Workflow-Fragment identifizieren. Die Iteration über die Liste von Dateipfaden und die Selektion des jeweiligen Dateipfades aus der Liste können über das *DIP* (siehe Kapitel 5.4) beschrieben werden. Die Liste der Dateipfade entspricht der Menge S. Man iteriert über diese Liste mittels der *forEach-Schleife*. Das über die Aktivität *Get_Path* selektierte Dateipfad aus der Liste entspricht der Menge T_i. Die TransferData Aktivität entspricht dem C2CP (siehe Kapitel 5.3). Dieses DMP entspricht der Anwendung der Operation op_i. Hierbei wird die jeweilige Quell-Datei aus der Liste im Workflow-Rechner in den Arbeitsverzeichniss im Pandas-Rechner kopiert. In diesem Fall betrachten wir das C2CP im DIP als ein Black-Box, wobei die Schleife zur Durchführung des Datentransfers in der übergeordneten forEach-Schleife sieben mal iteriert wird, d.h. so oft wie die Anzahl der Dateipfade in der Liste.

Parametrisierungsprozess

Als nächstes werden wir den Parametrisierungsprozess der identifizierten Patterns beschreiben. Bevor wir aber das tun möchten wir darauf hinweisen, dass wir für sämtliche Referenzen auf Dateien, ab sofort die im Kapitel 3.2 genannten *Data Container Reference Variables (DCVs)* verwenden. Der Grund ist, dass die aktuelle Implementierung des Pandas-Workflows (und von SIMPL) in Kürze auf diese Arten von Variablen geändert wird. Zur Identifikation der referenzierten Datencontainer im Workflow unterscheiden wir zwei Fälle:

1. Der Datencontainer ist im Resource Management registriert: In diesem Fall wird der Datencontainer in Worklow über einen Logical Container Name (siehe Kapitel 3.3), der auf den Datecontainer referenziert, im Resource Management eindeutig identifiziert. Jeder Datencontainer wird genau eine Datenquelle zugeordnet.
2. Der Datencontainer wird im Pattern direkt über den Local Container Identifier (siehe Kapitel 3.3) angegeben : Dann muss in der entsprechenden Data Container Reference

Variable zusätzlich noch die Datenquelle mit angegeben werden. Das ist der Fall im hier betrachten Beispiel bzw. das wird so angenommen.

Nachdem wir dies geklärt haben, werden wir mit dem Paramtrisieurungsprozess der DMPs fortfahren. Als erstes zeigen wir, welche Parameter für die einzelne DMPs festgelegt werden. Abbildung 6.4 illustriert dies.

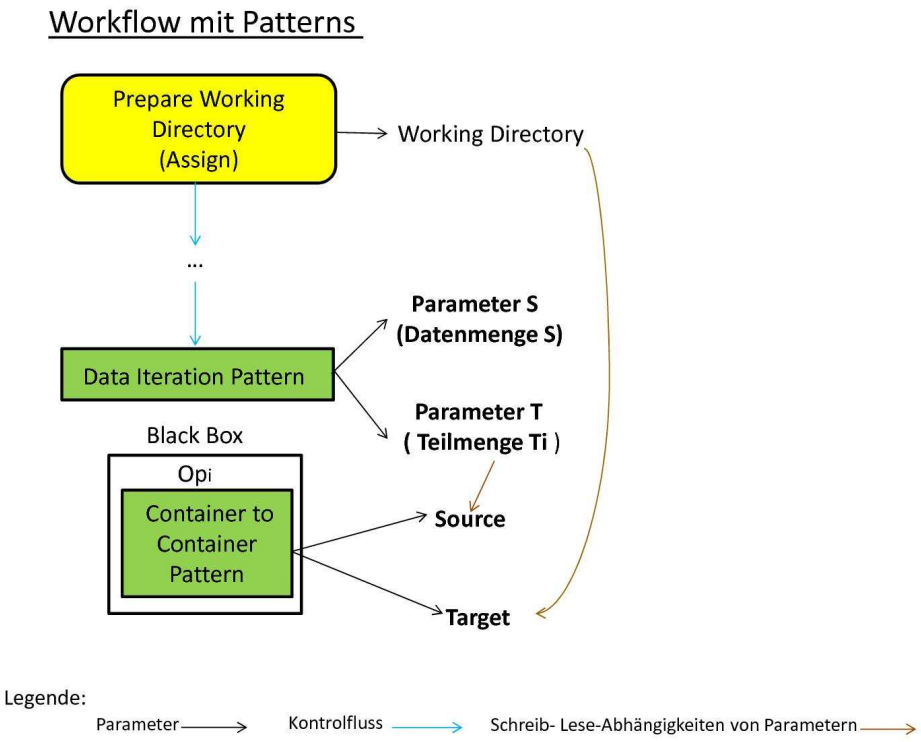


Abbildung 6.4: Parametrisierung der einzelnen Datenmanagementpatterns

Die Parameter sind erstmal allgemein beschrieben. Für das DIP wird der *Parameter S* festgelegt. Dieser Parameter entspricht in diesem Beispiel der Liste der Dateipfade. Der *Parameter T* entspricht der selektierten Quelle aus der Liste. Um den Datentransfer zwischen der Quell-Datei und der Ziel-Datei durchzuführen, muss das C2CP die Quelle und das Ziel kennen. Die Quell-Datei wird vom Parameter T ausgelesen und wird auf den Parameter *Source* kopiert. Die Ziel-Datei wird von der Variable "workingDirectory"der Assign-Aktivität *prepareWorkingDirectory* (siehe oben) ausgelesen und auf den Parameter *Target* kopiert. Für BPEL-Aktivitäten im Kontrolfluss des Workflow-Fragments müssen keine Parameter angegeben werden, da sie ausführbare Aktivitäten sind und keine DMPs.

Im nächsten Schritt müssen die Datentypen/Variabletypen der Parameterwerte der einzelnen Patern-Parameter bestimmt werden. Die Tabelle 6.1 illustriert dies. Der Wert des

Parameter	Parameterwerte
Parameter S	List of Data Container References
Parameter T	Data Container Reference
Source	Data Container Reference
Target	?Data Container Reference

Tabelle 6.1: Angabe der Variablentypen der Parameterwerte der jeweiligen Pattern-Parameter

Parameters S ist eine *Liste von DCRs*. Diese referenzieren auf die jeweilige Quelle-Datei in der Liste von DateipfadenPattern-. Für den Parameter T wird der Wert *Container Reference Variable* angegeben. Diese Variable referenziert auf die jeweilige Quell-Datei, die aus der Liste von Dateipfaden selektiert wird. Der Wert des Parameters Source ist eine *Container Reference Variable*. Sie referenziert auf die Quelle-Datei, deren Daten auf die Ziel-Datei, von der die Daten kopiert werden sollen. Der Wert des Parameters Target ist ebenfalls eine *Container Reference Variable*. Diese Variable referenziert auf die Ziel-Datei im Arbeitsverzeichnis des Pandas-Rechners, auf die die Daten kopiert werden.

Im Workflow-Fragment werden Platzhalter für die einzelne Parameterwerte definiert. Da im Workflow-Fragment solche Platzhalter stehen , ist das Workflow-Fragment noch nicht ausgeführt. Damit das Workflow-Fragment ausführbar gemacht werden kann, muss dieses Workflow Fragment zunächst identifiziert werden. Hierbei wird dem Workflow-Fragment ein eindeutiger Name vergeben. Weiterhin müssen die Platzhalter durch die entsprechende Variablen im Workflow-Fragment ersetzt werden.

Der Platzhalter Referenz List wird durch die Variable, die der Modellierer des Patterns beim Parameter S angegeben hat ersetzt, z.b "List". Der Platzhalter Container Reference wird von der Variablen die der Modellierer des Patterns beim Parameter T angegeben hat ersetzt, z.B. Input_Data_Path. Abbildung 6.5 zeigt wie die Platzhalter Für die Quelle (?sourceStatement), das Ziel(?targetStatement) und die jeweilige Datenressourcen (?Datasource und ?Datasink) durch die entsprechende Variablen ersetzt werden. Das Kopieren von Daten von der Quell-Datei auf die Ziel-Datei kann über eine *TransferData-Aktivität* ausgeführt werden. Zuerst muss er angegeben werden, von welcher Datei in welchem Rechner sollen die Daten kopiert werden. Für den Platzhalter ?sourceStatement muss also die Variable *Input_Data_Path* (Quell-Datei) und für den Platzhalter (?Datasource) die Variable *DataSourceLocal* (Workflow-Rechner) angegeben werden. Weiterhirn muss angegeben werden, auf welche Datei in welchem Rechnern sollen die Daten kopiert werden. Für die Platzhalter ?targetSource und ?Datasink sollen also die Variablen "workingDirectory"(Ziel-Datei) und DataSourceRemote (Panda-Rechner) angegeben werden. Damit wird die DataTransfer-Aktivität ausführbar. Aufgrund der Tatsache, dass die *Datenformate* der Dateien *identisch sind* kann sie somit die Daten aus der jeweiligen Quell-Datei aus der Liste auf die jeweilige Ziel-Datei kopieren.

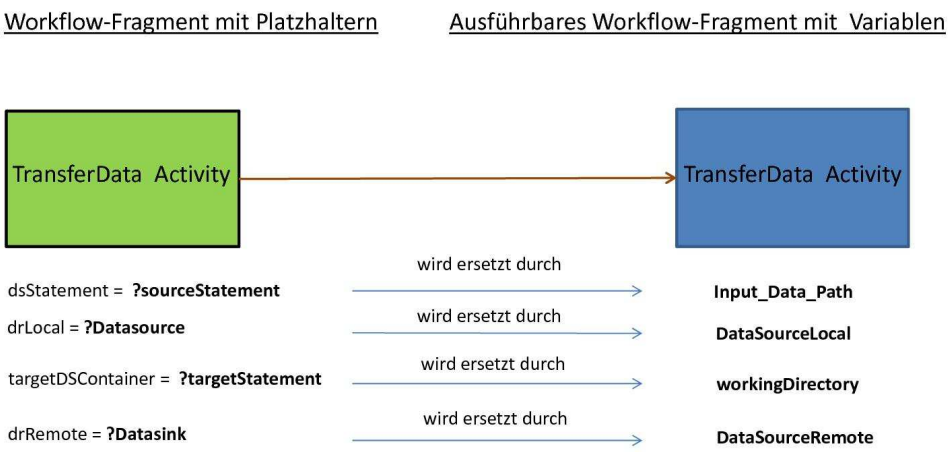


Abbildung 6.5: Ersetzen der festgelegten Platzhalter durch Variablen. Dadurch wird die TransferData-Aktivität ausführbar

Zum Schluss wollen wir zeigen wie die Datenquellen und die Datenformate der jeweiligen Datencontainer, also im Beispiel der jeweiligen Dateien, identifiziert werden. Dies kann man anhand der im Pattern angegebenen Datencontainer Referenz Variablen herausbekommen. Hierbei unterscheiden wir zwei Fälle:

1. Der Datencontainer ist im Resource Management registriert: In diesem Fall wird der Datencontainer in Workflow über einen Logical Container Name (siehe Kapitel 3.3), der auf den Datecontainer referenziert eindeutig identifiziert. Dann wird das Datenformat der von der Datenconatainer Referenz Variable referenzierten Datei eindeutig identifiziert, da ihr Datenformat auch im Resource Management registriert ist. Jedem Datencontainern wird im Resource Management genau eine Datenquelle zugeordnet. Dann wird die entsprechende Datenquelle auch identifiziert.
2. Der Datencontainer wird im Pattern direkt über den Local Container Identifier (siehe Kapitel 3.3) angegeben: Dann muss in der entsprechenden Data Container Reference Variable zusätzlich noch die Datenquelle mit angegeben werden. Da die Datencontainer Referenz Variable bereits einen lokalen Bezeichner enthält, muss wohl auch in

dieser Variable das Datenformat (das auch im Resource Management registiert ist) mit angegeben werden. .

Definition des Condition- und Action Part

Damit wir das parametrisierte DIP ausführbar machen, müssen wir es auf ein ausführbare Workflow Fragment abbilden. Dies geschieht über eine Tranformationsregel vom Typ 1 (siehe Kapitel 6.1). In diesem Fall wird die Operation im DIP, also das C2CP, als *Black Box* betrachtet, da die Abbildungsregel unabhängig von der Operation sein sollte. In einem solchen Fall muss allerdings rekursiv überprüft werden, ob sich in der Black-Box immer noch DMPs befinden. Im Folgenden wird der Condition- und Action Part dieser Transformationsregeln definiert.

Condition Part:

- Condition 1: Der Wert des Parameters S ist vom Typ *Reference List*.
- Condition 2: Der Wert des Parameters T ist eine *Container Reference Variable*.

Diese Voraussetzungen müssen erfüllt sein, damit das DIP auf ein ausführbares Workflow-Fragment transformiert werden kann. Im nächsten Schritt müssen wir den Action Part definieren. Hierbei wird das richtige Workflow-Fragment identifiziert und die Platzhalter durch die entsprechende Variablen ersetzt. *Action Part:*

- Action 1: Identifiziere das Workflow-Fragment preprocesing.
- Action 2: Der Platzhalter Referenz List wird durch die Variable, die der Modellierer des Patterns beim Parameter S angegeben hat ersetzt. Im Beispiel heißt diese Variable *List*.
- Action 3: Der Platzhalter Container Reference wird von der Variablen die der Modellierer des Patterns beim Parameter T angegeben hat ersetzt. Im Beisiel heißt diese Variable *Input_Data_Path*.

In einem zweiten Schritt müssen wir den Condition- und Action Part für das C2CP definieren. *Condition Part:*

- Condition 1: Der Wert des Parameters Source ist eine *Container Referenz Variable*.
- Condition 2: Der Wert des Parameters Target ist eine *Container Referenz Variable*.
- Condition 3: Die Datenformate der von den beiden Container Referenz Variablen referenzierten Dateien sind identisch.

Action Part:

- Action 1: Identifiziere das Workflow-Fragment Pandas.
- Action 2: Der Platzhalter ?sourceStatement wird von der Variablen *Input_Data_Path* ersetzt.
- Action 3: Der Platzhalter ?Datasource wird vom Platzhaltern *DataSourceLocal* ersetzt.

- Action 4: Der Platzhalter ?targetStatement wird von der Variablen *workingDirectory* ersetzt.
- Action 5: Der Platzhalter ?Datasink wird von der Variablen *DataSourceRemote* ersetzt.

6.2.2 Beispiel anhand der Pandas Postprocessing-Phase

Das zweite Beispiel, das wir vorstellen, wird aus der postprocessing-Phase der Pandas-Simulation entnommen. Dabei geht es um die Visualisierung der Endergebnisse der Simulation. Die Visualisierung wird von einem externen Werkzeug durchgeführt.

Funktionsweise und Identifizierung der Patterns

Auf der Abbildung 6.6 wird der Workflow zur Pandas Postprocessing-Phase dargestellt. Über die Aktivität *Save End State* wird das Endergebnis der Simulation auf einer Datei im

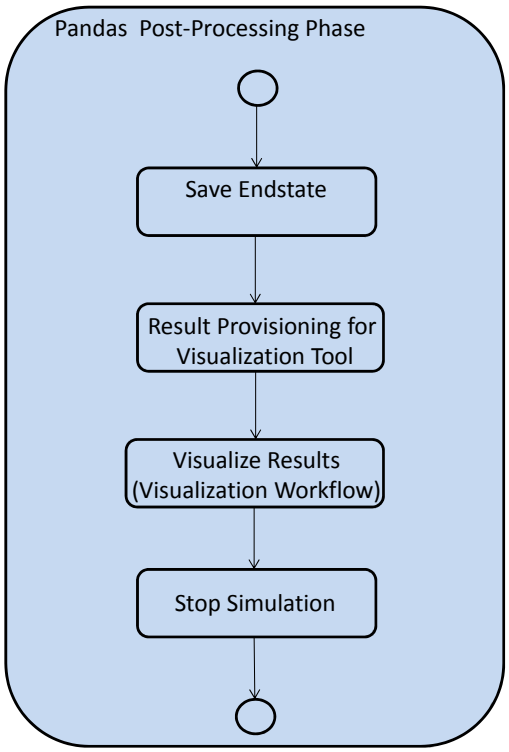


Abbildung 6.6: Workflow zur Pandas-postprocessing-Phase

Simulationsverzeichnis des Pandas-Rechners kopiert. Über die Aktivität *Result Provisioning for Visualization Tool* werden die Ergebnisdaten an das externe Visualisierungswerkzeug

auf dem Workflow-Rechner übergeben. Hierbei werden die Ergebnisdaten von einer Datei (tecplot_final.dat) im Arbeitsverzeichnis des Pandas-Rechners auf eine andere Datei(visualizationInput.vtk) auf den Workflow-Rechner kopiert. Wie sich aus den Endungen der Namen der beiden Dateien erkennen lässt, besitzen sie *unterschiedliche Datenformate*. Über die Aktivität *Visualize Results (Visualization Workflow)* wird die Visualisierung ausgeführt. Diese Aktivität entspricht einem anderen Workflow, der den Visualisierungsprozess Schritt für Schritt beschreibt [Jin09]. Die Aktivität *stop Simulation* beendet die Pandas-Simulation.

Im Rahmen dieses Beispiels werden wir beschreiben, wie der Datentransfer zur Übergabe der Ergebnisdaten zur Visualisierung der Simulation verläuft. Wie oben erwähnt findet dieser Datentransfer zwischen zwei Dateien mit unterschiedliche Datenformaten statt. Dies kann über ein *C2CP* beschrieben werden. Aufgrund der Tatsache, dass die Quell-und Ziel-Datei unterschiedlichen Datenformate besitzen, kann eine TransferData-Akrivität den Datentransfer alleine *noch nicht ausführen* (siehe Kapitel 6.2.1). Dafür muss vor der Datenübertragung eine *Datenformatkonvertierung* vorgenommen werden. Dazu braucht man ein *Python-Transformatiosskript*, das *tecp2vtk.py* heißt. Über dieses Transformationsskript wird das Datenformat der Datei *tecplot_final.dat* im Simulationsverzeichnis des Pandas-Rechners auf das für die Visualisierung benötigte Datenformat(*.vtk*) konvertiert. Anschließend wird eine Datei im Simulationsverzeichnis erzeugt, die das Format der Visualisierungs-Datei besitzt. Diese Datei heißt *feld.vtk*. Anschließend wird diese Datei in den Arbeitsverzeichnis des Workflow-Rechners kopiert. Diesen letzeten Transferschritt kann man wie im Kapitel 6.2.1 über ein *C2CP* beschreiben, da die dabei beteiligten Datenformate nach der Ausführung des Transformationsskripts wieder identisch sind. Die Datenfomatkonvertierung kann über ein *Datenformatkonvertierungspattern (DKP)* (siehe Kapitel 5.2) dargestellt werden.

Parametrisierungsprozess

Nachdem wir die relevanten Patterns identifiziert haben, werden wir mit dem Parametrisierungsprozess fortfahren. Wie in Kapitel 6.2.1 werden wir mit der Zuordnung der Parameter für die entsprechende Patterns beginnen. Abbildung 6.7 illustriert dies. Für das C2CP mit den unterschiedlichen Datenformaten werden die Parameter *Source* (tecplot_final.**dat**) und *Target* (visualizationInput.**vtk**) festgelegt. Aufgrund der Tatsache, dass die Datenformate der beiden Dateien unterschiedlich sind, kann eine *TransferData-Aktivität* den Datentransfer nicht ausführen. Damit der Datentransfer dennoch ermöglicht wird muss das C2CP über eine Anwendung einer Transformationsregel vom Typ auf andere DMPs transformiert werden. Darauf werden wir weiter unten eingehen. Zuerst wird das C2CP auf ein DKP, das das Datenformat **.tec** der Quell-Datei auf das für die Visualisierung benötigte Datenformat **.vtk** der Ziel-Datei konvertiert, danach auf ein C2CP, das den Datentransfer zwischen der Quell-Datei (feld.**vtk**) und der Ziel-Datei (visualizeInput.**vtk**) beschreibt. Für das DKP werden zwei Parameter festgelegt. Die Cintainer Reference Variable der Eingabe-Datei (tecplot_final.**dat**) wird vom C2CP mit den unterschiedlichen Datenformaten ausgelesen und auf den Parameter *Source* kopiert. Der Parameter $Target_1$ entspricht der vom DKP erzeugten Datei feld.**vtk** (Ausgabe-Datei). Die entsprechende Container Reference Variable wird vom C2CP mit den identischen Datenformaten ausgelesen und auf den Parameter $Source_1$ kopiert. Die Container Reference Variable der Ziel-Datei

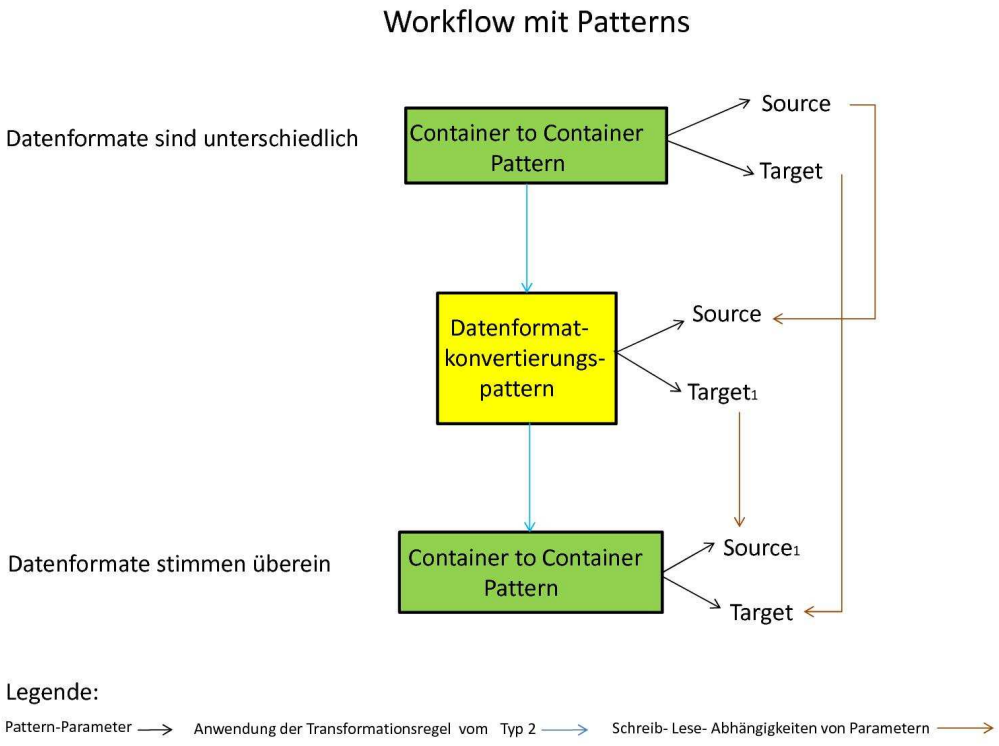


Abbildung 6.7: Parametrisierung der einzelnen Datenmanagementpatterns in der Postprocessing-Phase

(visualizeInput.**vtk**) wird vom C2CP mit den unterschiedlichen Datenformaten ausgelesen und auf den Parameter *Target* kopiert.

Da die Festlegung von Platzhaltern und das Ersetzen der Platzhalter durch die entsprechende Variablen für das C2CP mit den identischen Datenformaten bereits im Kapitel 6.2.1 gezeigt wurde, werden wir nicht nochmal darauf eingehen. An dieser Stelle werden wir das Festlegen der Platzhalter für die Parameter des DKPs und das Ersetzen dieser Platzhaltern durch die entsprechende Variablen beschreiben. Das DKP wird auf eine IssueCommand Aktivität (siehe Kapitel 3.2) abgebildet. Abbildung 6.8 illustriert dies. Als erstes muss der Rechner, auf den der DM-Befehl zum Aufruf des Transformationsskripts ausgeführt wird, angegeben werden. Der Platzhalter ?Datasource entspricht also der Datenressource, in dem sich das Transformationskript und seine Eingabe-und Ausgabedateien befinden. Wie diese Datenressource anhand der im Pattern angegebenen Datencontainer Referenz Variablen identifiziert wird, haben wir schon im Kapitel 6.2.1 besprochen. Weiterhin muss der Platzhalter für das *DM-Command* zum Aufruf dieses Transformationsskriptes angegeben werden. Der Platzhalter DM-Command enthält drei weitere Platzhalter. Als erstes muss

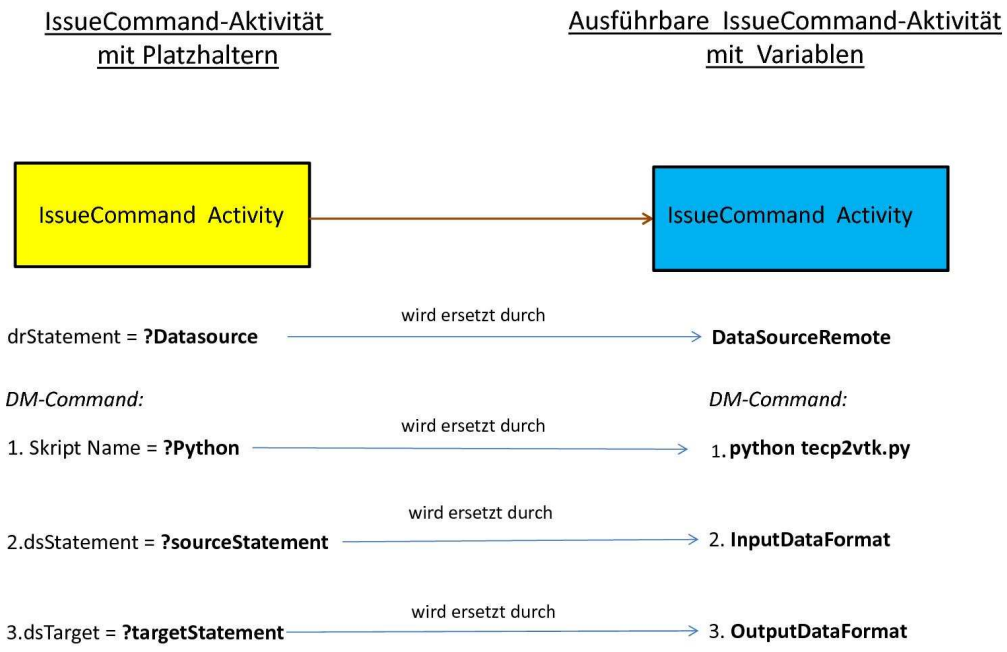


Abbildung 6.8: Ersetzen der Platzhalter durch die entsprechende Variablen. Dadurch wird die IssueCommand-Aktivität ausführbar

der Befehl zum Aufruf des Transformationsskriptes angegeben werden. Das geschieht über den Platzhalter *?Python*. Weiterhin müssen die Argumente zum Aufruf des Python-Skriptes festgelegt werden. Das erste Argument ist die Eingabe-Datei (tecplot_final.**dat**) und das zweite Argument die Ausgabe-Datei (feld.**vtk**). Der Platzhalter (*?sourceStatement*) entspricht der Eingabe-Datei und der Platzhalter (?targetStatement) der Ausgabe-Datei.

Die Datenformatkonvertierung kann über eine *IssueCommand-Akrivität* (siehe Kapitel 3.2) ausgeführt werden. Damit aber die Aktivität ausführbar wird, muss der Nutzer die entsprechende Variablen an der Stelle der entsprechenden Platzhalter angeben. Der Platzhalter ?Datasource wird durch die Variable *DataSourceRemote* ersetzt, also das Python-Skript wird im Pandas-Rechner aufgerufen. Weiterhin muss der Nutzer das entsprechende DM-Command festlegen. Der Platzhalter ?python wird durch den String *python tecp2vtk.py* ersetzt, also den konkreten Namen des Python-Skriptes. Weiterhin müssen die Eingabe- und die Ausgabe-Datei als Argumente für den Skript-Aufruf angegeben werden. Der Platzhalter ?statementSource wird durch die Variable *InputDataFormat* und der Platzhalter ?targetStatement durch die Variable *OutpatDataFormat* ersetzt.

Zum Schluss werden wir erklären, wie das Python-Skript als passender Dienst zur Durchführung der oben beschriebenen Datenformatkonvertierung identifiziert wird. Dafür werden die notwendige *Metadaten* im Ressource Management (siehe Kapitel 3.3) benötigt. Abbildung 6.9 zeigt die drei Objekte, die zur Identifikation des Python-Skriptes beteiligt sind. Zuerst müssen die beteiligte Datencontainer (Dateien) identifiziert werden. Wie im Kapitel

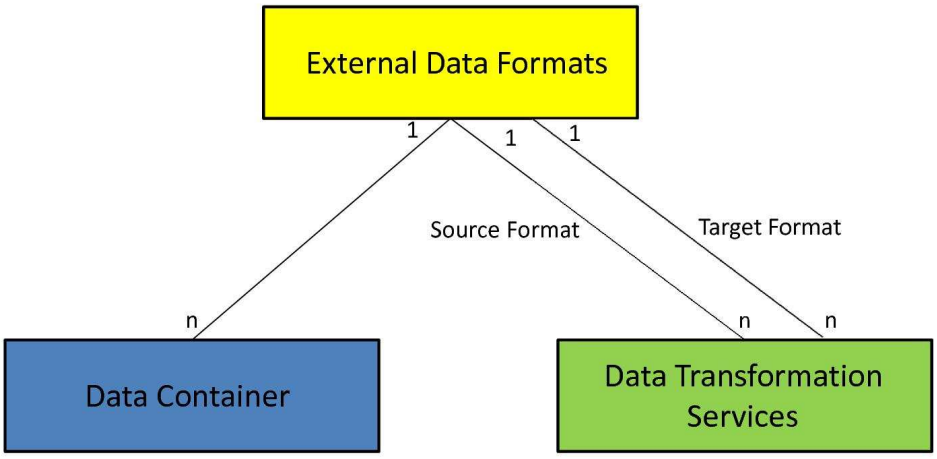


Abbildung 6.9: Metadaten und Metadaten-Objekte im Resource Management zur Datenformatkonvertierung

6.2.1 erläutert, unterscheiden wir zwei Fälle. Wenn die beiden Datencontainer im Data Container-Objekt des Resource Managements registriert sind, dann werden diese über einen Logical Container Name, der auf sie referenziert, eindeutig identifiziert. Wenn sie aber nicht registriert sind dann stehen die lokalen Bezeichner in den Datenconatiner Referenz Variablen. An dieser Stelle erweitern wir das Resource Management um ein weiteres Objekt. Im External Data Formats-Objekt sind alle externen Datenformate registriert. Anhand des Datenformats der identifizierten Quell-Datei (.tec) und des Datenformats der identifizierten Ziel-Datei (.vtk), wird der passende Dienst gesucht, der das Datenformatkonvertierung ausführen kann. Die Metadaten über *Data Transformation Services* sind im *Datatransformationservices*-Objekt registriert [Pie11]. Diesem Objekt werden folgende Attribute zugeordnet:

1. *id*: In diesem Attribut wird Data Transformation Service eine id vergeben.

2. *Name*: In diesem Attribut wird der Name des Transformation Service (tecTovtk) gespeichert.
3. *Source_Dataformat*: Referenziert aus das Datenformat eines Eingabe-Datebcontainers. Im Beispiel ist es das Datenformat (tec).
4. *Target_Dataformat*: Referenziert auf das das Datenformat des Ausgabe-Datencontainers. Im Beispiel ist es das Datenformat (vtk).
5. *Skript-Aufruf:* Beschreibt wie der Datenaufruf aussieht.

Nachdem der passende Service identifiziert wurde, wird er aus dem Resource Managemnet geladen und über die IssueCommand-Aktivität per SSH-Verbindung mit dem Pandas-Rechner aufgerufen.

Definition des Condition- und Action Part

An dieser Stelle werden wir beschreiben, wie die Abbildung der DM-Patterns auf ausführbare Workflow-Fragmente aussieht. Weiterhin werden wir die Condition- und Action-Part der entsprechenden Transormationsregeln erläutern. In diesem Beispiel wird das C2CP mit den unterschiedlichen Datenformaten rekursiv verfeinert, um es in einer solchen Form zu bringen, so dass eine direkte Transformation über einen Regeltyp-1-Ansatz realisierbar ist. Es wird also eine Regel vom Typ2 auf dieses Pattern angewendet (siehe Kapitel 6.1). Da es beim Datenkonvertierungspattern um ein feingranulares Pattern geht und weil eine IssueCommand-Aktivität (siehe Kapitel 3.2) das Transformationsskript starten kann, wird auf dieses Pattern eine Regel vom Typ 1 angewendet (siehe Kapitel 6.1). Das C2CP kann nach der Datenformatkonvertierung über eine Regel vom Typ 1 transformiert werden, da die Quell-und Ziel-Datei keine unterschiedlichen Datenformate mehr besitzen und weil eine TransferData-Aktivität den Datentransfer durchführen kann. Zum Schluss müssen wir den Condition und Action Part für die drei DMPs definieren. Für das C2CP mit den unterschiedlichen Datenformaten sieht die Definition des Condition- und Action Parts wie folgt aus:

Condition Part:

- Condition 1: Der Platzhalter des Parameters Source ist eine *Container Referenz Variable*.
- Condition 2: Der Platzhalter des Parameters Target ist eine *Container Referenz Variable*.
- Condition 3: Die Datenformate der beiden Datencontainern sind verschieden.

Action Part

- Action 1: Das Workflow-Fragment Transform wird richtig identifiziert.
- Action 2: Der Platzhalter ?statementSource wird von der Variablen *InputDataFormat* ersetzt.
- Action 3: Der Platzhalter ?Datasource wird vom Platzhaltern *DataSourceRemote* ersetzt.

Im nächsten Schritt werden wir den Condition und Action Part für das Datenkonvertierungspattern definieren.

Condition Part:

- Condition 1: Der Platzhalter des Parameters Source ist eine Container Referenz Variable.
- Condition 2: Der Platzhalter des Parameters $Target_1$ ist eine Container Referenz Variable.
- Condition 3: Die beide Datencontainer, die Datenformate und die zugehörigen Data Transformation Services sind im Resource Management registriert.

Action Part:

- Action 1: Das Workflow-Fragment Convert wird richtig identifiziert.
- Action 2: Der Platzhalter ?Datasource wird durch die Variable DataSource Remote ersetzt.
- Action 3: Der Platzhalter ?Python wird durch den String python tecp2vtk.py ersetzt.
- Action 4: Der Patzhalter ?statementSource wird durch die Variable InputDataFormat ersetzt.
- Action 5: der Platzhalter ?statementTarget wird durch die Variable OutputDataFormat ersetzt.

Zum Schluss werden wir den Condition-und Action Part für das C2CP mit den identischen Datenformaten definieren. Im nächsten Kapitel werden wir anhand der Definition der Kontrollstrategie und zwei Algorithmen zeigen, wie die im diesen Kapitel definierte Condition- und Action Parts der Transformationsregeln umgesetzt werden und wie die Transformation eines DMPs auf ein ausführbares Workflow-Fragment aussieht.

6.3 Definition der Kontrollstrategie

Im Rahmen dieses Kapitels werden wir zunächst die Definition der allgemeinen Konrollstrategie erläutern. Anschließend werden wir zwei Algorithmen vorstellen, die Die Definition der Kontrollstrategie und das Konzept der Pattern-Transformation umsetzen. Zum Schluss werden wir anhand der im Kapitel 6.2 vorgestellten Beispielen (DIP, C2CP, DKP) zeigen wie die Definition der Kontrollstrategie für diese DMPs aussieht.

6.3.1 Kontrollstrategie und Algorithmen zur Pattern-Transformation

Um die Transformation der Datenmanagementpatterns auf ausführbare Workflow-Fragmente zu realisieren, wird die Kontrollstrategie definiert.[Rei11] Wie im Kapitel 6.1 erwähnt bekommt der Pattern-Transformer einen Workflow mit Patterns und BPEL-Aktivitäten als Eingabe. Dieser Workflow kann als einen Graph dargestellt werden. Man traversiert durch diesen Graphen, um jedes besuchte Pattern durch ein ausführbares Workflow-Fragment zu ersetzen. Ein Beispiel eines solchen Graphen ist der Graph auf der Abbildung 6.3. Er repräsentiert die Zieltransformation. Die ganze *forEach-Schleife* entspricht dem *DIP* und die Aktivität *Transfer_Input_Files* dem *C2CP*. Da die BPEL-Aktivitäten ausführbar sind und keine DMPs, werden sie während der Traversierung ignoriert.

Über die Definition der Kontrollstrategie werden die passenden Regel für das jeweilige Eingabe-Pattern geliefert und die Reihenfolge der Überprüfung auf Anwendbarkeit der Regeln gesteuert. Anschließend wird es mit der Überprüfung der Regel auf Anwendbarkeit begonnen (Condition Part). Wenn der Condition Part erfüllt ist, dann wird die Regel auf das Pattern angewendet (Action Part). Die folgende Algorithmen illustrieren dieses Vorgehen. Die Algorithmen sind an [VSS$^+$ia] angelehnt und an das Verfahren der Pattern-Transformation entsprechend angepasst.

Algorithmus: 6.1 Traversierung durch den Workflow-Graphen vgl.[VSS$^+$ia]

```
Procedure TraverseWG(wg)
  while wg is not fully traversed do
   dmp ← getNextDMP(wg)
   if dmp ≠ null
   TransformDMP(dmp)
   end if
 end while
```

Algorithmus: 6.2 Pattern-Transformation vgl.[VSS$^+$ia]

```
Procedure TransformDMP(dmp)
 cs ← getControlstrategy(dmp)
 while cs is not finished do
     r ← getNextRule(cs)
     if r isApplicable(dmp,r) then
     wf ←applyrule(dmp,r)
      if r.Type=2 ∨ dmp.Type=DIP then
       traverseWG(wf)
      end if
       replace(dmp,wf)
       return
     end if
 end while
Escalation("There is no applicable rule")
```

Die Prozedur *TraverseWorkflow-Graph* nimmt als Eingabeparameter den zu transformierenden Workflow-Graphen *wg* und traversiert durch diesen Graphen über die Funktion *getNextDMP*. Für jeden Iterationschritt nimmt diese Funktion das nächste zu transformierende DMP aus dem Workflow-Graphen und liefert dieses, sofern es existiert, als Eingabeparameter an die *TransformDMP-Prozedur*. Diese Prozedur übernimmt die Transformation eines DMPs auf ein ausführbares Workflow-Fragment. Die Funktion *getControllStrategy* liefert die passende *Kontrollstrategie cs* für das eingegebene DMP. Die Funktion *getNextRule* liefert jeweils die nächste *Regel r* auf Basis der gefundenen Kontrollstrategie. Die Funktion *isApplicable* überprüft, ob die Regel r auf das DMP anwendbar ist, sie evaluiert also den Condition Part der Regel. Wenn die Regel r nicht anwendbar ist, dann wird es an die while-Schleife zurückgekehrt und mit der Überprüfung der nächsten Regel fortgesetzt. Wenn die Regel r anwendbar ist, dann wird sie über die Funktion *applyrule* auf das DMP angewendet, hier wird also er Action Part evaluiert. Als Ergebnis liefert sie das *Workflow-Fragment wf*. Was danach passiert, hängt davon ab, ob es sich bei der Regel r um einen Regeltyp 1 oder einen Regeltyp 2 handelt oder ob das Patern dmp ein DIP ist. Bei einem Regeltyp 2 haben wir in dem Workflow-Fragment wf erneut DMPs drin, d.h. wf ist noch nicht ausführbar. Wir führen daher rekursiv die Prozedur TraverseWG auf dieses Workflow-Fragment aus. Dadurch wird es dann rekursiv in ein ausführbares Workflow-Fragment transformiert, unter Umständen in mehreren Rekursionsschritten. Bei einem DIP, in dem die Operation op_i als Black-Box betrachtet wird (siehe Kapitel 6.2), wird die oben beschriebene Rekursion ebenfalls ausgeführt, da man nicht sicher sein kann, ob in dieser Black-Box auch Patterns enthalten sind oder nicht. Bei einem Regeltyp 1 bzw. nach der Rekursion bei einem Regeltyp2 oder einem DIP wird das DMP über die Funktion *replace* durch das Workflow-Fragment ersetzt. Anschließend wird die while-Schleife über die Funktion *return* unterbrochen. Damit wird auch die Prozedur TransformDMP beendet. Dadurch kommen wir wieder an die Prozedur TraverseWG, in der das nächste Pattern gefunden wird, wenn es noch eins gibt. Das gesamte Programm wird beendet, wenn alle DMPs im Workflow Graphen ein mal besucht und auf Anwendbarkeit überprüft wurden. Das Ergebnis des Programms ist der ausführbare Workflow-Graph.

Wenn alle Regeln r in der Kontrollstrategie cs auf das DMP nicht anwendbar sind, dann führen wir eine *Eskalation* ein, da wir keine Regel für das DMP gefunden haben. In einem solchen Fall können folgende Maßnahmen ergriffen werden:

- Der Nutzer kann ein Workflow-Fragment über die Verwendung eines Werkzeugs automatisch generieren.
- Der Nutzer kann sein eigenes Workflow-Fragment definieren.
- Man kann dem Nutzer Vorschläge für mögliche Workflow- Fragmente geben.

6.3.2 Anwendungsbeispiele für die beiden Algorithmen

Zum Schluss möchten wir anhand der DMPs aus den Kapiteln 6.2.1 und 6.2.2 Anwendungsbeispiele für die beiden Algorithmen angeben. Hierbei werden wir davon ausgehen, dass die

DMPs im Workflow-Graphen besucht wurden und dass die entsprechende Kontrollstrategie definiert wurde (siehe Kapitel 6.4).

Der Condition Part ist für das DIP erfüllt (siehe Kapitel 6.2.1). Es wird anschließend die Regel auf das DIP angewendet und das entsprechende Workflow-Fragment geliefert. Da die erste if-Bedienung erfüllt ist und weil die Operation (C2CP) im DIP als Blackbox betrachtet wird (siehe Kapitel 6.2.1), wird die Rekursion ausgeführt, da man nicht sicher sein kann, ob in dieser Black-Box Patterns vorhanden sind oder nicht. Dadurch kommen wir wieder an die Prozedur TraverseWG, in der das C2CP gefunden wird. Wenn der im Kapitel 6.2.1 definierte Condition Part für das C2CP erfüllt ist, dann wird die Regel auf das C2CP angewendet und das entsprechende Workflow-Fragment geliefert. Die zweite if-Bedienung ist aber in diesem Fall nicht erfüllt, da es in diesem Fall um eine Transformationsregel vom Typ 1 handelt (siehe Kapitel 6.2.1). Damit wird das C2CP durch das entsprechende Workflow-Fragment ersetzt. Dadurch kommen wir über die Funktion return wieder an die Prozedur TraverseWG. Danach kommt man über die Rekursion wieder an die Prozedur TansformDMP, in der das DIP durch das zuvor erzeugte Workflow-Fragment ersetzt wird. Nach diesem Schritt kommen wir an die Prozedur TraverseWG. Da der Baum gänzlich traversiert wurde wird das Programm beendet und das ganze Workflow-Fragment als Ergebnis ausgegeben.

Wenn der Condition Part für das C2CP mit den unterschiedlichen Datenformaten erfüllt ist, wird die Regel auf das C2CP angewendet und das entsprechende Workflow-Fragment geliefert. Da es sich im diesen Fall um eine Transformation vom Regeltyp-2 handelt, ist die if Bedienung erfüllt. Dadurch kommen wir über die Rekursion wieder an die Prozedur TraverseWG, wo das DKP gefunden wird. Der Condition Part ist für dieses Pattern ebenfalls erfüllt (siehe Kapitel 6.2.2). Anschließend wird die Regel auf das DKP angewendet und das entsprechende Workflow-Fragment geliefert. Da es sich in diesem Fall um einen Regeltyp 1 handelt, ist die zweite if-Bedienung nicht erfüllt. Damit wird das DKP durch das entsprechende Workflow-Fragment ersetzt. Dadurch kommen wir über die return Funktion wieder an die Prozedur TraverseWG, wo das C2CP gefunden wird. Der weitere Verlauf wurde im Beispiel zuvor erläutert. Daher werden wir nicht noch mal darauf eingehen. In diesem Fall wird die aus dem Kapitel 6.2 beschriebene Transformationsregel vom Typ 1 als erste genommen und überprüft, da diese schneller auf ein ausführbares Workflow-Fragment abgebildet werden könnte und weil es keiner andere Regeltyp 1 für das C2CP existiert, also es kann nur bei einem Datentransfer über das Kopieren von Daten von einer Datei auf eine andere Datei einen Regeltyp 1 für das C2CP geben.

6.4 Definition der Kontrollstrategie anhand von konkreten Beispielen

An dieser Stelle möchten wir zeigen, wie die Definition der einzelnen Kontrollstrategie für das DIP, das C2CP und das DKP aus dem Kapitel 6.2. aussehen. Für jeden Pattern-Typ gibt es genau eine Kontrollstrategie, die jeweils von der Operation getControlStrategy

(siehe Kapitel6.3.2) geladen wird. Für das DIP wird automatisch die aus dem Kapitel 6.2 beschriebene Transformationsregel vom Typ 1 als erste genommen und überprüft, da es keine andere Regel für das DIP existiert. Weiterhin muss die Kontrollstrategie für das C2CP definiert werden. Im Fall des C2CPs gibt es zwei Regel . Einen Regeltyp1 aus dem ersten Beispiel und einen Regeltyp2 aus dem zweiten Beispiel. Um DMPs möglichst früh auf ausführbare Workflow-Fragmente abzubilden, könnte es sinnvoll sein, die Typ 1-Regeln vor der Typ 2-Regeln auf Anwendbarkeit zu überprüfen [Rei11]. Daher wird die Regel aus dem ersten Beispiel als erste überprüft. Für das DKP wird automatisch die Regel vom Regeltyp 1 genommen da sie die einzige ist.

6.5 Architektur des Abbildungsmechanismus

Zur Umsetzung des im Rahmen dieser Arbeit vorgestellten Konzepts zur Pattern-Transformation haben wir eine Architektur erarbeitet. Auf Abbildung 6.10 ist diese Architektur zu sehen. Die Architektur besteht aus sieben Komponenten:

1. *Workflow Graph Traverser:* Traversiert durch den Workflow-Graphen und liefert das jeweils nächste DMP im Workflow-Graphen zur Pattern Transformer Engine. Die Traversierung durch den Workflow-graphen und die Lieferung des Patterns kann entweder zur Modellierungszeit- (Modeling Tool) oder zur Laufzeit stattfinden (Workflow-Engine) [Hum11].
2. *Ruleset and Control Strategies:* In dieser Komponente werden die Regelmenge und die Kontrollstrategie verwaltet.
3. *Pattern-Transformer Engine:* Diese Komponente stellt die zentrale Komponente für die Transformation der Patterns in ausführbare Workflow Fragmente dar und steuert diese Transformation. Sie besteht aus zwei weiteren Komponenten. Die *Control Strategy Engine* iteriert über die Kontrollstrategie eines bestimmten Patterns bzw. liest die jeweils nächste Regel aus. Die*Rule Engine* enthält zwei weitere Komponenten. Die *Condition Evaluation Engine* überprüft die Transformationsregeln auf Anwendbarkeit. Die *Action Part Evaluation Engine* wendet die Transformationsregeln auf die DMPs an und. Im Anschluss an die Regelanwendung leitet die Pattern Transformer Engine das Workflow-Fragment an den Workflow Graph Traverser (nur bei Rekursion, also Regeltyp2 oder DIP)oder an die Fragment Composition-Komponente weiter.
4. *SIMPL Ressource Management:* In dieser Komponente werden die Objekte und die zugehörigen Metadaten verwaltet. Diese Matadaten werden zur Überprüfung der Transformationsegeln auf Anwendbarkeit miteinbezogen.
5. *Workflow Fragment Management:* Die ausführbare Vorlagen für Workflow-Fragmente werden in dieser Komponenten verwaltet und bereitgestellt. Die Workflow Fragmente befinden sich in der WF-Library.

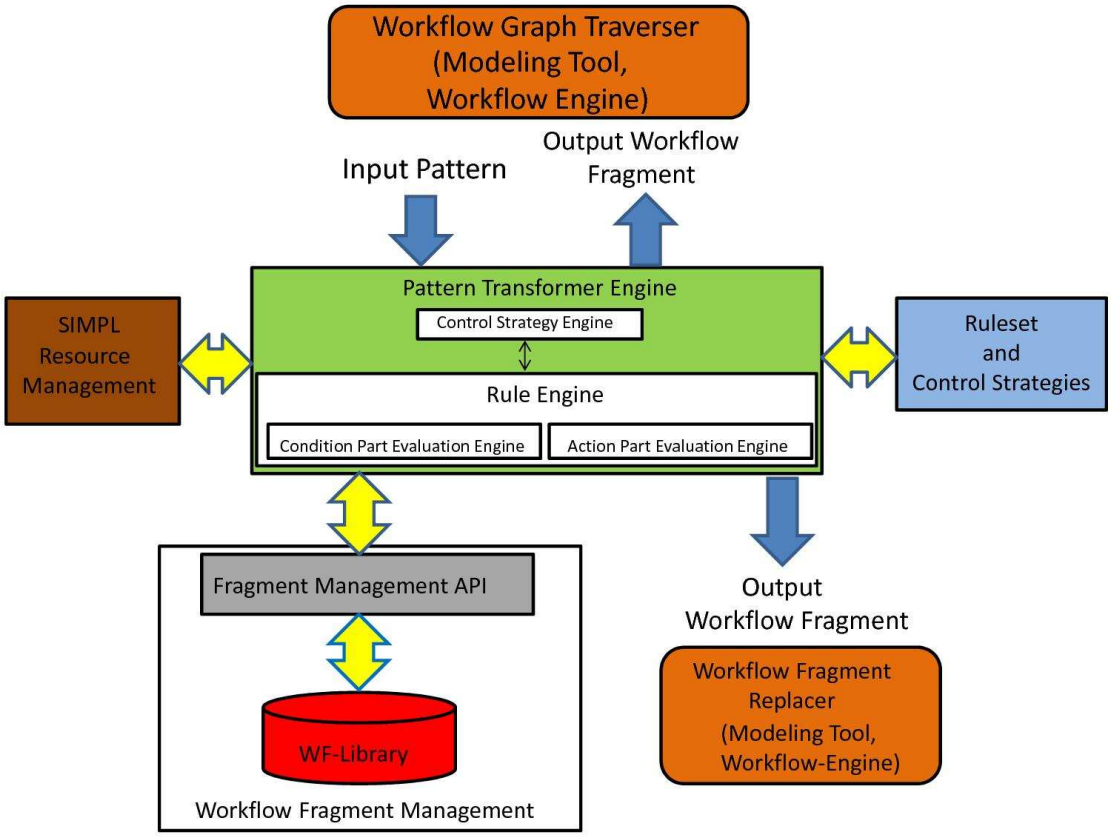


Abbildung 6.10: Architektur des Abbildungsmechanismus

6. *WF Management API:* Diese Komponente stellt die Schnittelle zur WF-Library mit den anderen Komponenten dar, insbesondere mit der Pattern Transformer Engine, die nach Workflow Fragmenten bzw. deren Metadaten anfragen kann.

7. *Workflow Fragment Replacer:* Diese Komponente führt das Ersetzen der Patterns durch die Workflow-Fragmente durch.

Die einzelnen Komponenten der Architektur setzen die Baumtraversierung, die Überprüfung der Transformationsregeln auf Anwendbarkeit und die Pattern-Transformation anhand der im Kapitel 6.3 vorgestellten Algorithmen um. Weiterhin wird die Zusammensetzung der DMPs implementiert. Der Algorithmus *Traverse-WG* wird von der Komponentne Workflow Graph Traverser implementiert. Das Traversieren des Workflow-Graphen kann entweder zur Modellierungszeit oder zur Laufzeit erfolgen [Hum11]. Im ersten Fall wird der Algorithmus *TraverseWG* (siehe Kapitel 6.3) in der GUI, z.B.Eclipse BPEL Designer, implementiert. Zunächst werden die Workflows mit den DMPs vom Modellerer erstellt. Anschließend wird der Algorithmus TraverseWG auf den jeweiligen Workflow-Graphen angewendet.

Hierbei wird das jeweils nächste besuchte DMP dem Pattern-Tramsformer geliefert. Im zweiten Fall wird der Algorithmus in der Execution-Engine, z.B.Apache-ODE, implementiert. Hierzu werden zunächst die in der Modellierungszeit erstellten Workflow-Graphen in der Execution-Engine deployt. Anschließend wird durch den Workflow-Graphen navigiert. Wenn der Navigator das jeweilige nächste DMP im Workflow-Graphen besucht, dann liefert er es dem Pattern-Transformer.

Im nächsten Schritt fragt die Pattern Transformer Engine in der Ruleset and Control Strategies-Komponente nach der passenden Kontrollstrategie, in der die Regelmenge und ihre Reihenfolge der Überprüfung auf Anwendbarkeit enthalten bzw. beschrieben ist. Die Komponente Ruleset and Controllstrategies implementiert dabei die Funktion *getControlstrategy(dmp)* (siehe Kapitel 6.4) des Algorithmus *TransformDMP*. Hierbei wird die Regelmenge ausgegeben, die Kontrollstrategie geladen und das Ergebnis der Pattern Transformer Engine geliefert.

Die Control Strategy Engine in der Pattern Transformer Engine implementiert die Funktion getNextRule(cs). Sie iteriert über die gelieferte Kontrollstrategie, liest die jeweils nächste Regel aus und leitet sie an die Rule Engine weiter. Die *Condition Part Evaluation Engine* der Rule Engine implementiert die Funktion *isApplicable(dmp,r)*. Hierbei wird die gelieferte Regel anhand der im Condition Part festgelegten Voraussetzungen (siehe Kapiteln 6.1 und 6.2) auf Anwendbarkeit überprüft. Zu diesem Zweck fragt der Pattern-Transformer im SIMPL Resource Management und ggf. auch in der Komponente Workflow Fragment Management, falls man Metadaten zu den Fragmenten braucht, nach relevanten Metadaten, z.B. Metadaten über Datencontainer, Datenressourcen, externe Datenformate und Data Transformation Services (siehe Kapitel 6.2), ab.

Im nächsten Schritt findet die Evaluierung des Action Part statt. Zunächst vergibt die Action Part Evaluation Engine jedem einzelnen Workflow-Fragment eine ID. Danach fragt die Action Part Evaluation Engine in der Workflow Management API nach dem passenden Workflow Fragment ab. Die Workflow Management API leitet diese Anfrage an die WF-Library der Workflow Fragment Management Komponente weiter. In der WF-Library werden die ausführbare Vorlagen für Workflow-Fragmente gespeichert [SKK+11]. Gegen diese Library können verschiedene Queries ausgeführt werden, wie z.B das Auslesen, das Suchen, die Bereitstellung und das Extrahieren von Fragmenten. Dadurch können sie wiederverwendet und in anderen Prozessen integriert werden. Nachdem die passende Vorlage in der WF-Library gefunden wurde, wird die Transformationsregel auf das entsprechende Fragment angewendet. Die Action Part Evaluation Engine implementiert dabei die Funktion *applyrule(dmp,r)*. Falls in diesem Fragment erneut Patterns enthalten sind, dann wird es an die Workflow Graph Traverser Komponente weitergeleitet.

Bei einem Regeltyp 1 bzw. nach der Rekursion bei einem Regeltyp2 oder einem DIP wird das DMP durch das Workflow-Fragment ersetzt. Hierbei implementiert die Komponente Workflow Fragment Replacer die Funktion *replace(dmp,wf*. Das Ersetzen der Patterns durch die ausführbare Workflow-Fragmente kann entweder zur Modellierungs- oder zur Laufzeit erfolgen vgl.[Hum11]. Im ersten Fall wird die Funktion replace(dmp,wf)

in der GUI implementiert. Im zweiten Fall werden die Patterns durch die ausführbare Workflow-Fragmente in der Workflow-Engine ersetzt.

Die im Kapitel 6.3 beschriebene Eskalation, wenn für ein bestimmtes Pattern keine Regel gefunden wurde, wird in dieser Architektur nicht berücksichtigt.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde eine Abstraktionsunterstützung für die vereinfachte Definition des Datenmanagements in Simulationsworkflows entwickelt und vorgestellt. Diese Abstraktionsunterstützung basiert auf Datenmanagementpatterns, die innerhalb von Workflow-Modelle verwendet und modelliert werden können. Anhand von konkreten Anwendungsszenarien haben wir solche Datenmanagementpatterns herausgearbeitet und identifiziert. Die ETL-Patterns/Operationen, das Data Transfer and Transformation Pattern und seine Unterklassen, nämlich das Container to Container Pattern, das Data Split Pattern und das Data Merge Pattern, und schließlich das Data Iteration Pattern wurden formalisiert und erläutert. Zur Klassifizierung und Beschreibung dieser Datenmanagementpatterns wurde eine Hierarchie, die aus fünf verschiedenen Schichten besteht, vorgestellt.

Um die definierten Datenmanagementpatterns ausführbar zu machen, wurde ein Konzept erarbeitet. Dieses Konzept basiert auf der Anwendung von Transformationsregeln. Jede Transformationsregel setzt sich aus einem Condition und einem Action Part zusammen. Im Condition Part werden die Bedingungen beschrieben, unter denen eine gewisse Transformationsregel auf ein DMP angewendet werden darf. Diese Bedingungen hängen zum einen mit den Parameterwerten des zu transformierenden Patterns und zum anderen mit den Metadaten, die die beteiligten Datenressourcen und Workflow-Fragmente beschreiben, zusammen. Im Action Part einer Regel wird festgelegt, wie dieses DMP auf ein ausführbares Workflow-Fragment abgebildet wird, wenn die Regel auf das DMP angewendet wird. Hierbei gibt es zwei Typen von Transformationsplänen. Der Regeltyp 1 und der Regeltyp 2. Über die Definition der Kontrollstrategie werden die passenden Regel für das jeweilige Eingabe-Pattern geliefert und die Reihenfolge der Überprüfung auf Anwendbarkeit der Regeln gesteuert. Zur Umsetzung unseres Konzepts haben wir eine Architektur erarbeitet und vorgestellt.

Eine solche Datenmangementabstraktion erlaubt die Vereinfachnung der Definition des Datenmnagements von Simulationsanwendungen mit einem großen Datenaufkommen in verteilten und heterogenen Umgebungen [RRS+ny]. Darüber hinaus unterstützt sie die Wissenschaftler bei der Definition des Datenmanagements für ihre Simulationsanwendungen, indem sie ihnen die parametrisierte, ausführbare Vorlagen für Workflow-Fragmente zur Verfügung stellt. Dadurch müssen sich die Wissenschaftler mit komplexen Datenmanagement- und Datenbereitstellungsaufgaben nicht befassen.

Ausblick

Diese Diplomarbeit kann als Basis für die Anfertigung weiterer Arbeiten verwendet werden. Folgende Punkte können z.B bearbeitet werden:

- Man kann zusätzliche Datenmanagementpatterns für weitere Simulationsanwendungen definieren. Zur Abbildung der Paterns auf ausführbare Workflow-Fragmente kann das im Rahmen dieser Arbeit vorgestelltes Konzept verwendet und ggf.erweitert werden.
- Der Fall des Auftritts von Komplikationen bei der Transformation der Datenmanagementpatterns wurde in dieser Diplomarbeit sehr grob behandelt. Man kann die einzelnen Fälle näher untersuchen und eventuelle Lösungsvorschläge für diese einzelne Fälle geben.
- Die Architektur des Abbildungsmechanismus kann detailliert werden. Einzelne Teile der Architektur des Abbildungsmechanismus können implementiert werden.

Literaturverzeichnis

[ADR06] A.Akram, D.Meridith, R.Allan. Evaluation of BPEL to Scientific Workflows. In: Proc. of the sixth International Symposium of Cluster Computing and the Grid, 2006. Vol.1, pp.269-274. (Zitiert auf den Seiten 16 und 29)

[A.So6] A.Slominski. *Adapting BPEL to Scientific Workflows*. In I.Taylor and E.Deelman and D.B.Gannon Workflows for e-Science, 2006. (Zitiert auf Seite 16)

[BHH+10] D. Brüderle, W. Hüttig, M. Hahn, M. Schneidt, R. Rehn, F. Zoabi, X. T. (Ed.). Begriffslexikon des Studienprojekts SIMPL, Universität Stuttgart, 2010. (Zitiert auf Seite 45)

[BIS+06] B.Ludäscher, I.Altindas, S.Bowers, J.Cummings, T.Critchlow. *Scientific Proces Automation and Workflow Management*. Chapman and Hall/CRC Computational Science, 1 edition, 2006. (Zitiert auf den Seiten 17 und 20)

[Coa95] W. M. Coalition. The Workflow Reference Modell, 1995. Http://www.wfmc.org/standards/docs/tc003v11.pdf. (Zitiert auf Seite 12)

[Cod70] E. Codd. *A Relational Model for Data for Large Shared Data Banks*. Comm.ACM 13:6, 1970. (Zitiert auf den Seiten 47, 49, 50 und 51)

[CTE09] C.Herath, T.Gunarathne, E.Chinthaka. Experience with Adapting a WS-BPEL Runtime for e-Science. In Proc. of the fifth Grid Computing Enviroment Workshop, New York, USA, 2009. (Zitiert auf Seite 16)

[Dor11] R. Dormien. *Service-Bus-Erweiterung um Pandas-basierte Simulationen in Workflows zu nutzen*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf den Seiten 17, 25, 35 und 69)

[EC08] E.Deelman, C.Chervenak. Data Management Challenges of Data-Intensive Scientific Workflows. In: Proc. of the International Symposium of Cluster Computing and the Grid, Washington DC, 2008. (Zitiert auf Seite 7)

[EUF09] H. Eberle, T. Unger, F.Leymann. *Process Fragments. On the move to Meaningful Internet Systems*. Springer, 2009. (Zitiert auf Seite 31)

[Gil07] Gil.Y. Examining the Challenges of Scientific Workflows. In: IEEE Computer, 2007. Vol.40. (Zitiert auf Seite 7)

[Hum11] A. Hummel. *Ausführung von Workflow-Fragmenten in BPEL*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf den Seiten 31, 86, 87 und 88)

[Jin09] Z. Jing. *Workflow getriebene Visualisierung*. Master's thesis, Universität Stuttgart, 2009. (Zitiert auf Seite 77)

[KSK+11] K.Gorlach, M. Sonntag, D. Karastoyanova, F.Leymann, M. Reiter. *Conventional Workflow Technology for Scientific Simulation*. Springer Verlag, 2011. (Zitiert auf den Seiten 5, 20, 21 und 22)

[LR00] F. Leymann, Dieter Roller. *Produktion Workflow-Concepts and Techniques*. Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2000. (Zitiert auf den Seiten 5, 7, 11, 12, 13 und 15)

[Mü10] C. Marian Müller. *Develpoment of an integrated Database Architekture for a Runtime Enviroment for Simulation Workflows*. Master's thesis, Universität Stuttgart, 2010. Http:// elib.uni.stuttgart.de/opus/volltexte/2010/5232/pdf/DIP_2984.pdf. (Zitiert auf den Seiten 5, 16, 17, 23, 35 und 36)

[OAS] OASIS. Web Services Business Process Execution Language Version 2.0, Oasis Standard. Http:// www/docs.osis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. (Zitiert auf den Seiten 14 und 70)

[P.H90] P.Humphreys. Computer Simulations. In: Proc. of the Biennial Meeting of the Philsophy of Science Association, 1990. Vol.2, pp. 497-506. (Zitiert auf Seite 16)

[Pie11] H. A. Pietranek. *Erweiterung von SIMPL und BPEL-DM zur Unterstützung weiterer Datenquellen*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf Seite 80)

[P.S03] M. P.Sherwood. A general purpose implementation of QM/MM approach and its application to problems in catalysis. JOURNAL OF MOLECULAR STRUCTURE-THEOCHEM, 2003. (Zitiert auf Seite 35)

[rai] Pandas-COUPLED FEM SOLVER, Manual of the Demo Version. http://www.get-Pandas.com. (Zitiert auf Seite 67)

[Rei11] P. Reimann. SimTech Milestone Report: Data Provisioning for Scientific Workflows. Insitut für Parallele und Verteilte Systeme, Universität Stuttgart, April, 2011. (Zitiert auf den Seiten 5, 63, 65, 83 und 86)

[Rem11] S. Remppis. *Ausführung einer Modellreduktion für Simulationen auf Basis der Workflow-Technologie*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf den Seiten 5, 35 und 41)

[RJDN09] R.Barga, J.Jackson, D.Guo, N.Gautam. The Trident Scientific Workflow Bench. In: Proc. of the IEEE Fourth Inter.Conf. on eScience, 2009. Pp. 317-318. (Zitiert auf Seite 24)

[RM11] P. Reimann, B. Mitschang. Data Provisioning for Scientific Workflows. Poster-Präsentation auf dem vierten SimTech Status Seminar, Bad Boll, Deutschland, 2011. (Zitiert auf den Seiten 5, 45, 46, 63 und 64)

[RRS+ny] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, F. Leymann. SIMPL-A Framework for Accesing External Data in Simulation Workflows. 14.Fachtagung des GI-Fachbereichs "Datenbanken und Informationssystemen" ,(DBIS), Proceedings, März, 2004, Kaiserslautern, Germany. (Zitiert auf den Seiten 5, 7, 19, 26, 27, 28, 31, 33, 34, 35, 37, 45, 46 und 91)

[RSM] P. Reimann, H. Schwarz, B. Mitschang. Data Provisioning Techniques for Simulation Workflows. Unvferöffentlichter Bericht des Instituts für Parallele und Verteilte Systeme, Universität Stuttgart. (Zitiert auf den Seiten 35, 54, 55 und 58)

[SCF+05] S.Weerawarana, F. Curbera, F.Leymann, T.Storey, D. Ferguson. *Web Services Platform Architekture: SOAP, WSDL, WS-Policy, WS-Addressing and more*. Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2005. (Zitiert auf den Seiten 9 und 11)

[S.H05] S.Hartmann. The world as a Process: Simulations in the Natural and Social Sciences, 2005. Http://phil-sci-archive.pitt.edu/archive/00002412/01/Simulations.pdf. (Zitiert auf Seite 17)

[SKK+11] D. Schumm, D. Karastoyanova, O. Kopp, F. Leymann, M. Sonntag, S. Strauch. Process Fragments Libraries for Easier and Faster Development of Process-based Applications. http://www.iaas.uni-stuttgart.de/institut/mitarbeiter/schumm/ART-2011-022011. (Zitiert auf Seite 88)

[TDG07] I. Taylor, E. Deelman, D. Gannon. *Workflows for e-Science-Scientific Workflows for Grids*. Springer,London,UK, 2007. (Zitiert auf den Seiten 7, 9 und 15)

[VsRM08] M. Vrohvnik, H. schwarz, S. Radeschutz, B. Mitschang. An Overview of SQL Support in Workflow Produkts. In: Proc. of the IEEE 24th Int. Conf. on Data Engineering ICDE, 2008. (Zitiert auf Seite 19)

[VSS+ia] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Meier, T. Kraft. An Approach to Optimize Data Processing in Business Processes. In: Proc. of the 33rd International Conference on Very Large Data Bases (VLDB),, Septemeber, 2007, Vienna, Austria. (Zitiert auf Seite 83)

[W3C07] W3C. SOAP Version 1.2, 2007. Http:// www.w3.org/TR/soap/. (Zitiert auf Seite 10)

[Wag11] F. Wagner. *Nutzung einer integrierten Datenbank zur effizienten Ausführung von Workflows*. Master's thesis, Universität Stuttgart, 2011. (Zitiert auf den Seiten 10, 15 und 19)

[ZTZ06] O. Zienkiewicz, R. Taylor, J. Zhu. *The Finite Element Method-It's Basis and Fundamentals*. Butterworth-Heinemann, 6 edition, 2006. (Zitiert auf Seite 17)

Alle URLs wurden zuletzt am 16.02.2012 geprüft.

Erklärung

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

(Stavros Aristidou)