Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 121

Dynamische Teilausführung von Mashup Plans zur Modellierungszeit

Nikolai Neugebauer

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl. Inf. Michael Behringer, Dipl. Inf. Pascal Hirmer
Beginn am:	18. Juli 2016
Beendet am:	17. Januar 2017
CR-Nummer:	C.1.3, D.2.11, D.2.12

Kurzfassung

Data Mashups gewinnen immer mehr Bedeutung, da heute in vielen Bereichen Daten ausgewertet werden müssen. Data Mashup Tools eigenen sich sehr gut für die Datenauswertung, da sie zumeist auch von Domänenexperten, die nicht programmieren können, bedient werden können.

FlexMash ist eine Data Mashup Plattform, die domänenunabhängig funktioniert. In dieser Arbeit wird FlexMash erweitert, sodass Mashups teilweie ausgeführt werden können. Damit kann die Wartezeit für den User verringert und somit die Usability erhöht werden. Außerdem können so unnötige Ausführen von Operationen und damit Rechenkapazitäten eingespart werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Daten spielen heute in fast allen Bereichen eine immer größer werdende Rolle. Insbesondere in Firmen werden heute unter dem Buzzword 'BigData' immer mehr Daten erfasst. Eine reine Erfassung von Daten ist jedoch nicht ausreichend. Daten müssen auch ausgewertet und interpretiert werden. Daten können allerdings nur dann ausgewertet und interpretiert werden, wenn sie einem Experten für die entsprechenden Daten in geeigneter Form präsentiert werden.

Hierfür ist in der Regel eine Filterung und Aufbereitung der gesammelten Daten notwendig. Die Aufbereitung großer Datenmengen ist ohne Programme nicht mehr machbar. Domänenexperten, die die Daten auswerten können, haben aber oftmals keine Programmierkenntnisse und können sich daher keine Programme schreiben, die die Daten aufbereiten. Jede erdenkliche Aufbereitung an die IT Abteilung auszulagern ist sehr teuer und daher oftmals nicht möglich. Es bedarf folglich Wege, um die Datenaufbereitung für Domänenexperten einfach zu machen. [Nai15]

Data Mashup Tools setzen an diesem Punkt an. In einer einfachen GUI soll der Domänenexperte verschiedene Datenquellen kombinieren, filtern und auswerten können. [DM14]

FlexMash [HM16; Hir+15] ist ein solches Data Mashup Tool, das domänenunabhängig eingesetzt werden kann. Außerdem ist FlexMash dank einer Drag-and-Drop GUI auch für Nicht IT-Experten einfach zu bedienen.

In dieser Arbeit soll die Usability von FlexMash weiter erhöht werden, indem Mashups teilweise ausgeführt werden.

1.1 Gliederung

Die Arbeit gliedert sich weiterhin wie folgt.

Kapitel 2 – Grundlagen stellt Technologien vor, die dieser Arbeit beziehungsweise FlexMash zu Grunde liegen.

Kapitel 3 – Verwandte Arbeiten präsentiert Arbeiten, deren Konzepte für die teilweise Ausführung von Mashup Plans verwendet werden könnten. Außerdem wird FlexMash vorgestellt.

Kapitel 4 – Anforderungen listet die Anforderungen auf, die in FlexMash implementiert werden sollen.

Kapitel 5 – Konzepte diskutiert, wie die Anforderungen aus dem vorherigen Kapitel umgesetzt werden könnten.

Kapitel 6 – Implementierung zeigt auf, wie die Konzepte in FlexMash realisiert wurden.

Kapitel 7 – Beispielszenario erläutert ein Beispiel, wofür FlexMash verwendet werden könnte.

Kapitel 8 – Zusammenfassung gibt die Ergebnisse dieser Arbeit zusammengefasst wieder.

Kapitel 9 – Ausblick gibt einen Ausblick, wie FlexMash basierend auf dieser Arbeit weiter erweitert und verbessert werden könnte.

1.2 Aufgabenstellung

FlexMash übersetzt die Mashup Plans in BPEL Workflows. [Hir+15] Bei der Generierung eines Mashups wird jeweils der komplette BPEL Workflow ausgeführt. Dies ist problematisch, da so schon kleine Änderungen an einzelnen Operationen dazu führen, dass der komplette Workflow neu ausgeführt wird, obwohl die Zwischenergebnisse von vielen Operationen identisch sind. Folge dieser kompletten Ausführung ist ein Berechnungsmehraufwand. Außerdem leidet auch die Usability, da die Ausführung so deutlich länger dauert.

In dieser Arbeit sollen Konzepte für die teilweise Ausführung von BPEL Workflows recherchiert und erarbeitet werden. Außerdem sollen diese Konzepte in FlexMash integriert werden, sodass die Usability von FlexMash verbessert werden kann.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt, auf denen diese Arbeit basiert.

2.1 Mashups

In diesem Abschnitt werden Mashups vorgestellt.

Mashups sind Anwendungen, die aus wiederverwendbaren Daten, wiederverwendbarer Anwendungslogik und wiederverwendbaren Userinterfaces zusammengesetzt werden. [DM14]

Mashups bestehen aus Mashup Komponenten und der Mashup Logik. [DM14]

Eine Mashup Komponente kann ein Datensatz, Anwendungslogik oder ein Userinterface sein. Sie kann wiederverwendet werden und ist entweder lokal oder remote verfügbar. [DM14]

Die Mashup Logik ist die interne Operationslogik des Mashups. Sie definiert die Komponenten Auswahl, den Kontroll- und Datenfluss sowie die Datentransformationen. [DM14]

Mashups werden aus drei Perspektiven betrachtet und können nach diesen eingeteilt werden. Abbildung 2.1 zeigt die drei Perspektiven auf. [DM14]

Die Composition Perspektive gibt an, welche Art von Komponenten zu einer neuen Anwendung zusammengesetzt werden. Dadurch entstehen drei Mashup

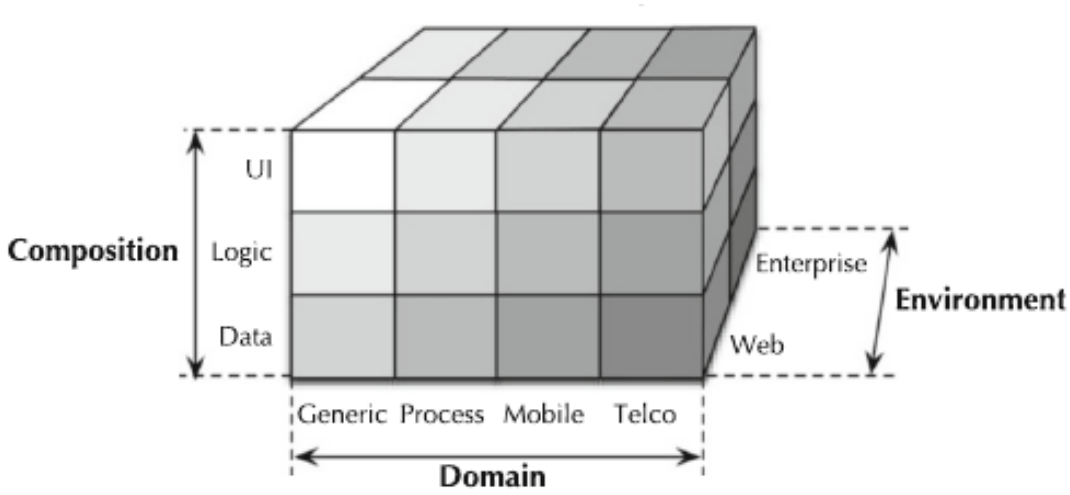

Abbildung 2.1: Drei Perspektiven auf Mashups [DM14]

Typen: Data Mashups, Logik Mashups und UI Mashups. Hybride Mashups kombinieren die drei genannten Typen beliebig. [DM14]

Die Domain Perspektive spezifiziert den Zweck des Mashups. Sie gibt an, welche Funktionalität das Mashup haben soll. [DM14]

Die Environment Perspektive gibt an in welchem Umfeld das Mashup verwendet werden soll. Web Mashups werden daher oft auch Consumer Mashups genannt, allerdings wählen Florian Daniel und Maristella Matera den Begriff Web Mashup, da die Ausführungsumgebung für sie relevanter ist, als der Anwender. [DM14]

Mashup Komponenten

Die Komponenten aus denen ein Mashup zusammengesetzt werden kann, können nach verschiedenen Kriterien sortiert werden. [DM14] Einige davon werden im folgenden vorgestellt.

Komponenten Typ

Zunächst gibt es die Unterscheidung nach dem Komponenten Typ. Hierbei wird unterschieden, was bei der Komponente wiederverwendet werden kann. [DM14]

- **Daten Komponenten** dienen dazu auf Daten zuzugreifen [DM14]
- **Logik Komponenten** bieten gekapselte Funktionalität zum Beispiel in Form von wiederverwendbaren Algorithmen [DM14]
- **User Interface Komponenten** haben ein eigenes Userinterface mit kompletter Anwendungslogik und Daten im Hintergrund. Das Userinterface kann mit anderen Userinterface Komponenten kombiniert werden. [DM14]

Ausführungsort

Eine weitere Kategorisierungsmöglichkeit ist der Ausführungsort der Komponente. Die Ausführung kann dabei entweder lokal beim Nutzer oder remote auf einem Serer stattfinden. [DM14]

Aufrufmethode

Mashup Komponenten können auch danach unterschieden werden, wie sie aufgerufen werden. Möglichkeiten sind:

- Local procedure calls
- Remote procedure calls
- Nachrichten basierte Aufrufe
- Event basierte Aufrufe [DM14]

Außerdem können die Komponenten Aufrufe synchron, oder asynchron sein. Bei synchronen Aufrufen warten nachfolgende Komponenten darauf, dass die vorherigen Komponenten im Daten- oder Kontrollfluss fertig ausgeführt sind. Bei asynchronen Aufrufen nicht. [DM14]

2.1.1 Mashup Kategorisierung

Auch Mashups können nach verschiedenen Kategorien sortiert werden.

Mashup Typen

Mashups können in vier Typen eingeteilt werden. [DM14]

Data Mashups arbeiten mit Daten. Sie verarbeiten Daten von verschiedenen Datenquellen, und führen darauf Operationen für die Aufbereitung und Darstellung der Daten aus. [DM14]

Data Mashups werden im Kapitel 2.2 ausführlicher behandelt.

Logic Mashups kombinieren Komponenten auf der Ebene der Anwendungslogik. [DM14]

Userinterface Mashups kombinieren verschiedene native Userinterfaces zu einem integrierten Userinterface. [DM14]

Hybride Mashups kombinieren alle drei anderen Mashup Typen in einem Mashup. [DM14]

Ausführungsort

Eine weitere Möglichkeit Mashups zu kategorisieren ist der Ausführungsort der Mashups. Dabei werden drei Möglichkeiten unterschieden:

- Clientseitige Ausführung
- Serverseitige Ausführung
- Client- und serverseitige Ausführung [DM14]

Integration

Eine wichtige Methode, um Mashups zu kategorisieren ist die Art der Integration, also wie die Komponenten zusammengesetzt werden. [DM14]

Userinterface basierte Integration kann nur bei Userinterface Komponenten angewendet werden. Hierbei werden die verschiedenen Userinterfaces der Komponenten nebeneinander in derselben Ansicht dargestellt. [DM14]

Orchestrierte Integration kann für alle Komponenten angewendet werden. Hierfür wird eine zentrale Composition Logik benötigt, die die einzelnen Mashup Komponenten ausführt und mit den einzelnen Komponenten kommuniziert. [DM14]

Datenaustausch

Die Art des Datenaustausches zwischen den Mashup Komponenten ist eine weitere Art, um Mashups zu kategorisieren. [DM14]

Die einfachste Möglichkeit ist der **direkte Austausch** zwischen den Komponenten. Hierfür wird keine Integrationslogik benötigt. [DM14]

Beim **Blackboard approach** werden die Daten von der Integrationslogik by-value an die Komponenten übergeben. Die Daten werden gelesen, wenn

Komponenten aufgerufen werden und gespeichert, wenn deren Ausgabe verarbeitet wird. [DM14]

Gemeinsamer Speicher ist ein typischer Ansatz für verteilte Systeme. Alle Komponenten nutzen hierbei denselben Speicher. Es wird also call-by-reference implementiert. [DM14]

Mediatorbasierter Datenaustausch erfordert, dass die Integrationslogik die Daten nicht nur temporär speichert, sondern auch bearbeitet. [DM14]

2.2 Data Mashups

Data Mashups sind eine spezielle Form von Mashups. Der Zweck von Data Mashups ist es die Daten aus verschiedenen Datenquellen zu kombinieren, die Daten aufzubereiten und darzustellen. [DM14]

Dabei gibt es fünf Hauptoperationen.

Datenzugriff: Die erste und wichtigste Operation in einem Data Mashup ist der Datenzugriff. Daten können dabei aus beliebigen Datenquellen bezogen werden. Dies können klassische Datenquellen wie Datenbanken sein, aber genauso gut auch RSS-Feeds, Webseiten, E-Mails oder soziale Netzwerke. [DM14]

Dateninterpretation: Nachdem die Daten geladen wurden, müssen sie interpretiert werden. Das heißt die Daten müssen vom Format der Datenquelle in ein Format gebracht werden, in dem die Daten bearbeitet werden können. [DM14]

Data mediation: Wenn die Daten verfügbar sind, kann es nötig sein mediation Operationen durchzuführen. Dabei werden die Daten in eine homogene Datenstruktur mit festgelegter Semantik gebracht. [DM14]

Entity resolution: Zusammengehörende Daten müssen zusammengeführt werden. Hierzu gehören zum Beispiel Join Operationen. Bei Data Mashups werden jedoch oftmals verschiedene Datenquellen mit unterschiedlichen Semantiken, Bezeichnern etc. zusammengeführt. Daher ist die Entity resolution oftmals eine komplizierte Aufgabe. [DM14]

Datamanipulation: Zusätzlich zum Zusammenführen der Daten, müssen Daten oftmals auch bearbeitet werden. Hauptsächlich können das Filteroperationen, Projektionen oder löschen von Daten sein. [DM14]

2.3 Mashup Tools

Mashup Tools oder Mashup Plattformen dienen dazu die Entwicklung der Mashup Logik, also der internen Abläufe innerhalb eines Mashups zu vereinfachen. Mashup Tools bieten hierfür häufig eine GUI, die eine modellbasierte Entwicklung von Mashups unterstützt. Zu einem Mashup Tool gehört in der Regel auch eine Ausführungsumgebung, in der die modellierten Mashups ausgeführt werden können. [DM14]

2.3.1 Nutzen von Mashup Tools

Es gibt verschiedene Anwendungsfälle in denen Mashup Tools von Nutzen sind. Die genauen Ziele müssen je nach Anwendungsfall definiert werden.

Vereinfachung: Durch ein Modellierungs ool kann die low-level Implementierung weg abstrahiert werden. Dies führt zu einer signifikanten Vereinfachung bei der Erstellung von Mashups. [DM14]

Erreichen von Anwendern ohne Programmierkenntnisse: Durch eine grafische Modellierung wird keine direkte Programmierung mehr benötigt, beziehungsweise sie wird stark vereinfacht. Somit können auch Anwender ohne Programmierkenntnisse Mashups erstellen. [DM14]

Steigern der Produktivität: Low-level Implementierungen sind immer mit Zeit und Kosten verbunden. Kann aus einem grafischen Modell automatisiert ein Mashup generiert werden, so steigert dies die Produktivität. [DM14]

2.3.2 Kategorisierung von Mashup Tools

Es gibt verschiedene Kategorien, nach denen Mashup Tools eingeteilt werden können. Ausgewählte Kategorisierungen werden im Folgenden vorgestellt.

Spezifizität: Ein Mashup Tool kann entweder für einen spezifischen Anwendungsfall erstellt worden sein, oder es ist generisch für viele verschiedene Anwendungsfälle einsetzbar. [ANP12]

Zielgruppe: Vom Programmierer bis zum absoluten nicht Programmierer gibt es viele Abstufungen. Je nach Komplexität der Modellierung richtet sich ein Mashup Tool an eine bestimmte Zielgruppe. [ANP12]

Automatisierungsgrad: Der Automatisierungsgrad gibt an, wie viel von der Mashuperstellung das Tool dem Anwender abnimmt. [ANP12]

Liveness: die Liveness gibt an zu welchem Grad das Mashup Tool Mashups in Echtzeit generieren kann. Die Abstufungen gehen hierbei von Level 1 bis 4. Auf Level 1 ist dabei gar keine Ausführung möglich. Auf Level 4 sind Änderungen während der Ausführung des Mashups möglich. [ANP12]

Interaktionsmöglichkeiten: Die Interaktionsmöglichkeiten geben an, womit die Modellierung erfolgt. Abstufungen sind hierbei zum Beispiel eine textbasierte Modellierungssprache, eine visuelle Sprache, What-You-See-Is-What-You-Get. [ANP12]

2.3.3 Endanwenderfreundliche Mashup Tools

Mashup Tools sind letztlich Composition Tools. Diese sollen nach [Cas11] einfach und mächtig sein. Außerdem sollen sie die benötigte Prozesslogik auf ein Mindestmaß reduzieren. Weiterhin soll die Komplexität innerhalb der Komponenten und nicht innerhalb der Composition Logik liegen. Zusätzlich muss die Konfiguration von Datenmappings automatisch erfolgen und nicht durch XPath oder ähnliches. Abschließend soll ein Composition Tool eine Schritt für Schritt Entwicklung unterstützen sowie die erwarteten Ergebnisse simulieren. [Cas11]

Die Schritt-für-Schritt Entwicklung wird auch inkrementelle Entwicklung genannt. [RI06]

Optimalerweise unterstützen Mashup Tools What-you-see-is-what-you-get. [AP13]

2.4 Web Services

Ein Web Service ist ein self-describing und self-contained Softwaremodul, das über ein Netzwerk erreichbar ist. [Pap08]

Web Services sollen das Problem starrer Implementierungen mit fest vordefinierten Abhängigkeiten lösen. Web Services sollen außerdem dynamisch miteinander kombiniert werden können. [Pap08]

2.4.1 Web Services Technology Stack

Der vollständige Web services technology stack ist in Abbildung 2.2 dargestellt.

Um Webservices zu verwenden werden jedoch nur SOAP, WSDL und für die Webservice Discovery UDDI benötigt. [Pap08]

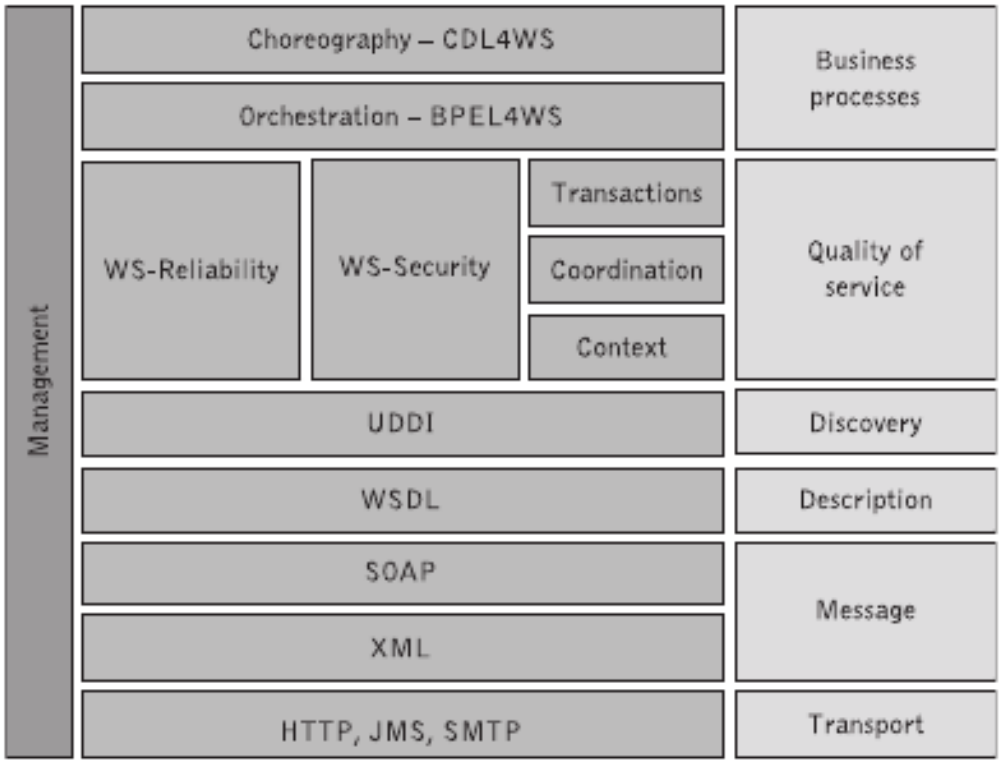

Abbildung 2.2: Web services technology stack [Pap08]

SOAP steht für Simple Object Access Protocol. Dabei handelt es sich um ein XML basiertes Nachrichtenaustauschprotokoll. Webservices nutzen SOAP Nachrichten, um miteinander zu kommunizieren. [Pap08]

WSDL steht für Web Service Description Language. WSDL ist eine XML basierte Sprache um Daten und Operationen eines Web Services zu beschreiben. Das WSDL Dokument definiert, wie ein Webservice verwendet werden muss. [Pap08]

2.5 BPEL

BPEL steht für Business Process Execution Language. BPEL ist eine Sprache, um Workflows zu beschreiben. Diese Workflows bestehen aus Web Services, die durch Daten- und Kontrollflüsse miteinander verbunden sind. [Pap08]

3 Verwandte Arbeiten

3.1 FlexMash

Diese Arbeit ist eine Erweiterung der Data Mashup Plattform FlexMash. [HM16; Hir+15] Daher wird in diesem Kapitel FlexMash vorgestellt.

FlexMash ist ein domänenunabhängiges Data Mashup Tool. Es besteht aus drei Komponenten:

- FlexMash GUI: Modellierungs Tool im Browser
- FlexMash Server: Runtime Environment für Mashups
- Repository für Operationen

Mashup Plan erstellen

In der FlexMash GUI kann der Domänenexperte einen sogenannten Mashup Plan erstellen (Abbildung 3.1). Ein Mashup Plan besteht aus Operationen, sowie Verbindungen zwischen den Operationen. [Hir+15]

Hierfür greift die GUI auf das Repository für Operationen zu. Im Repository können IT Experten Operationen anlegen, die sie entwickelt haben. Jede Operation ist dabei ein Webservice. In FlexMash gibt es zwei verschiedene Arten von Operationen: Datenquellen Operationen (englisch Datasource Descriptions) und Datenverarbeitungsoperationen (englisch Data Processing Descriptions). [Hir+15]

Die Verbindungen zwischen den Operationen stellen den Datenfluss dar.

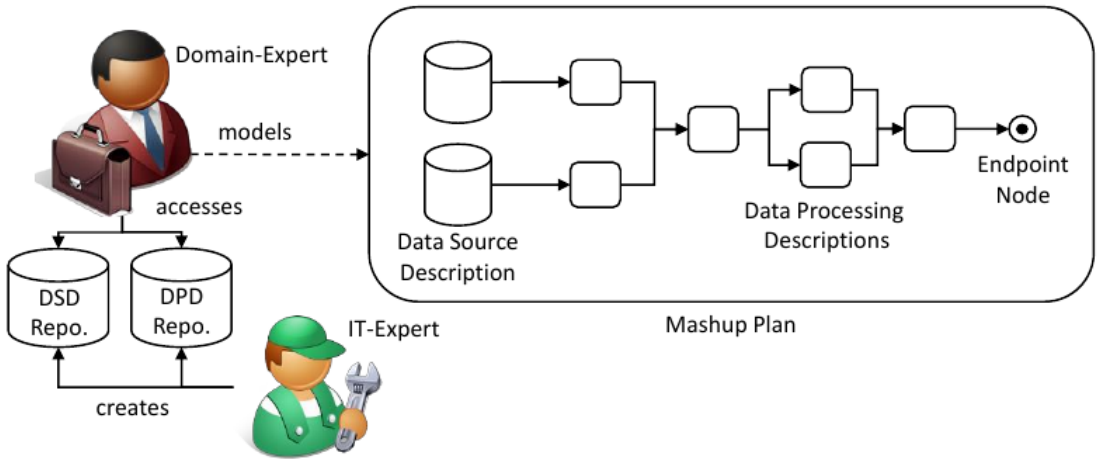

Abbildung 3.1: Modellierung eines Mashup Plans in FlexMash

Mashup Plan ausführen

Soll ein Mashup Plan ausgeführt werden, so wird der Mashup Plan an den Server gesendet. Dieser transformiert den Mashup Plan in einen BPEL Workflow, deployed diesen in einer Workflow Engine und führt ihn aus. [Hir+15]

Dieser Ablauf ist in Abbildung 3.2 dargestellt.

3.2 Smart re-runs

Im Scientific Workflow Management System Kepler werden sogenannte smart re-runs vorgestellt. Dies sind erneute Ausführungen eines Workflows, wobei sich nur einzelne Parameter von Operationen oder Rollen geändert haben. [ABJF06; Lud+06]

Bei einem sogenannten smart re-run werden die Teile des Workflows, deren Daten aus den Logs der vorherigen Ausführung verwendet werden können,

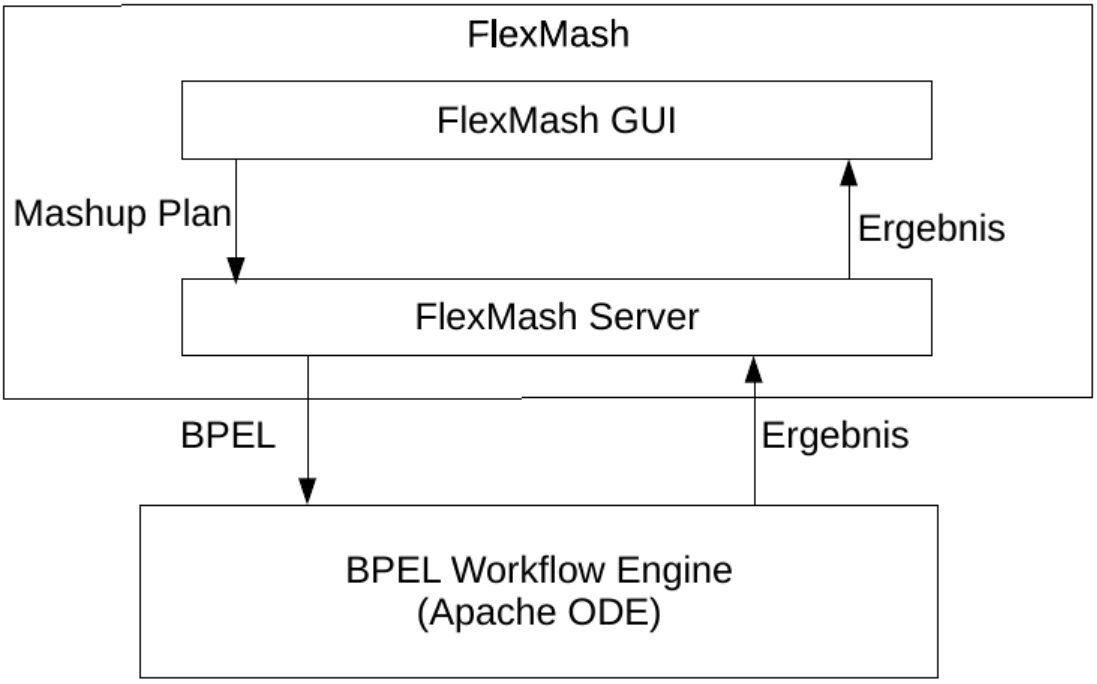

Abbildung 3.2: Ausführung eines Mashup Plans in FlexMash

durch eine andere Rolle ersetzt. Diese Rolle liest die Logs der vorherigen Ausführung ein und streamt die Daten an der benötigten Stelle in den Workflow. [ABJF06]

3.3 Model-as-you-go

Model-as-you-go ist ein Ansatz für Scientific Workflows, der eine erneute Ausführung von Workflowteilen unterstützt. Die Konzepte wurden für Apache ODE, eine BPEL Workflow Engine umgesetzt. [SK11; SK12; SK13]

Mit Model-as-you-go ist es möglich einen Workflow teilweise zurückzusetzen, sodass er ab einer ausgewählten Aktivität erneut ausgeführt werden kann. Hierfür wird die Workflow Instanz in der Workflow Engine wiederhergestellt. Die Aktivitäten, die der ausgewählten Startaktivität folgen werden mittels eines Compensation Handlers rückgängig gemacht. Danach kann der Workflow ab der ausgewählten Instanz neu ausgeführt werden. [SK11; SK12; SK13]

Ein zweites vorgestelltes Verfahren startet eine neue Workflow Instanz. Diese Workflow Instanz wird mit Hilfe des Audit Trails der vorherigen Ausführung auf den Stand vor der Startaktivität versetzt. Das Verwenden des Audit Trails ist nötig, um die Daten der nicht ausgeführten Aktivitäten zu erhalten. Damit ist die Workflow Instanz auf dem Stand, als wären die Aktivitäten vor der gewählten Startaktivität ausgeführt worden und die Workflow Instanz kann normal ausgeführt werden. [SK11; SK12; SK13]

4 Anforderungen

Die Kernanforderung dieser Arbeit ist die Verbesserung der Usability von FlexMash durch die Teilausführung von Mashup Plans. In diesem Kapitel werden die hierfür nötigen Änderungen am FlexMash Server vorgestellt. Außedem wird die Integration der Teilausführung in die Mashup Plattform diskutiert. Hieraus leiten sich die in 4.5, 4.6 und 4.7 vorgestellten Anforderungen ab.

Die Vorteile der Änderungen werden anhand eines schematischen Beispiel Mashup Plans (Abbildung 4.1) aufgezeigt. Der Beispiel Mashup Plan besteht ausfünf Operationen Operation 1 – Operation 5). Die Operationen senden ihre Ergebnisse jeweils als Nachricht (Nachricht 1 – 4) an die nachfolgende Operation. Operation 5 hat das Ergebnis des Mashups als Ausgabe.

4.1 Teilausführung von Mashup Plans

Die Kernanforderung ist die Teilausführung von Mashup Plans. Ändert ein Anwender die Konfiguration einer Operation, so muss derzeit der gesamte Mashup

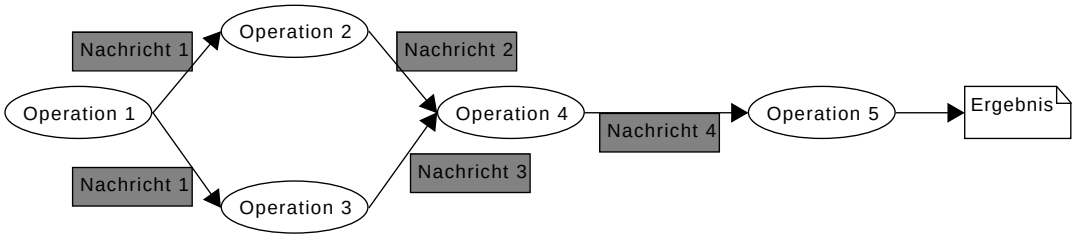

Abbildung 4.1: Schematischer Beispiel Mashup Plan

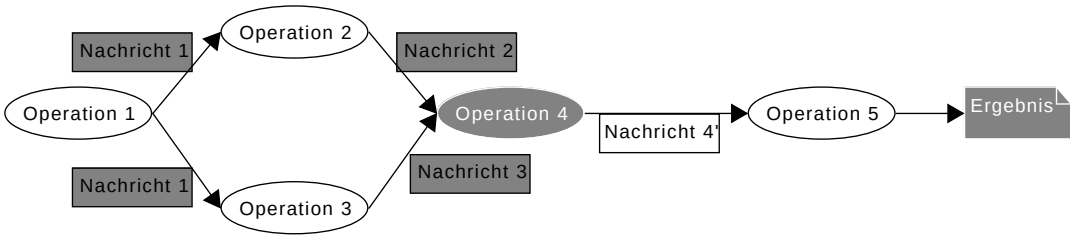

Abbildung 4.2: Beispiel Mashup Plan mit geänderter Konfiguration

Plan neu ausgeführt werden. Dies erzeugt unnötigen Berechnungsaufwand, da sich erst aber der geänderten Operation die Ergebnisse ändern. Insbesondere bei großen Datenmengen führt dies zu Usability Einschränkungen, da der User länger auf das Ergebnis warten muss.

Der FlexMash Server soll erkennen, welche Operationen geändert wurden und nur die Operationen ausführen, deren Ergebnis sich geändert hat.

Als Beispiel wurde in Abbildung 4.2 die Konfiguration von Operation 4 geändert. Dadurch ändert sich die Nachricht 4, die an Operation 5 versendet wird und somit auch das Ergebnis von Operation 5 und des gesamten Mashup Plans. Es sollen also die Operationen 4 und 5 neu ausgeführt werden, jedoch nicht die Operationen 1 bis 3.

4.2 Zwischenergebnisse weiterverarbeiten

Die Operationen eines Mashup Plans verarbeiten die Ergebnisse der vorherigen Operationen weiter. Wird ein Mashup Plan nicht von Beginn an ausgeführt, stehen die Ergebnisse der vorherigen Operationen nicht zur Verfügung. Um dennoch die Anforderung aus 4.1 Teilausführung von Mashup Plans umsetzen zu können, muss auf die Ergebnisse vorheriger Operationen auf andere Weise zugegriffen werden können. Außerdem müssen diese Zwischenergebnisse als Eingabedaten für nachfolgende Operationen verwendet werden.

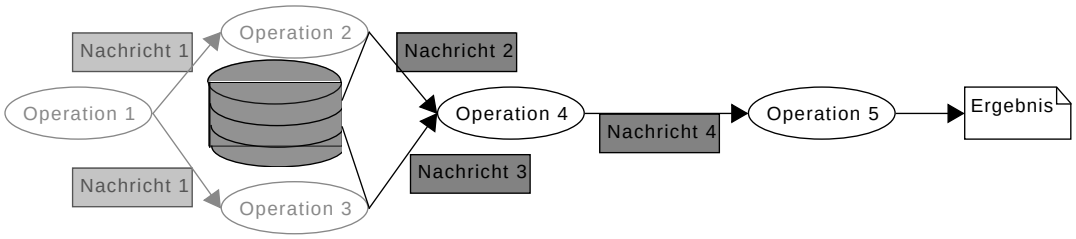

Abbildung 4.3: Beispiel Mashup Plan mit Zwischenergebnissen aus Datenbank

Der FlexMash Server muss also so erweitert werden, dass auf die Ergebnisse jeder einzelnen Operation zugegriffen werden kann. Außerdem muss FlexMash die Operationen so konfigurieren, dass diese Eingabedaten einer vorherigen Ausführung verwenden, statt von der vorherigen Operation.

Zur Veranschaulichung: In Abbildung 4.3 wurde wieder die Konfiguration von Operation 4 geändert. Die Operationen 4 und 5 müssen also neu ausgeführt werden, die Ergebnisse der Operationen 1 bis 3 sind bereits vorhanden. Die Eingabedaten von Operation 4 stammen dieses Mal also nicht von den Operationen 2 und 3 direkt, sondern aus einer Datenbank für vorherige Ergebnisse der Operationen 2 beziehungsweise 3.

4.3 Erkennen veralteter Daten

In 4.2 Zwischenergebnisse weiterverarbeiten wird erklärt, warum Daten von vorherigen Ausführungen verwendet werden sollen. Oftmals sind Daten jedoch dynamisch und verändern sich von Zeit zu Zeit. Werden immer die Daten einer vorherigen Ausführung verwendet, so kann es sein, dass die Daten veraltet sind, da sich die Daten in der Originalquelle inzwischen verändert haben.

Um dies zu verhindern muss FlexMash veraltete Daten erkennen und gegebenenfalls Operationen neu ausführen, obwohl Ergebnisse für die Operationen vorliegen.

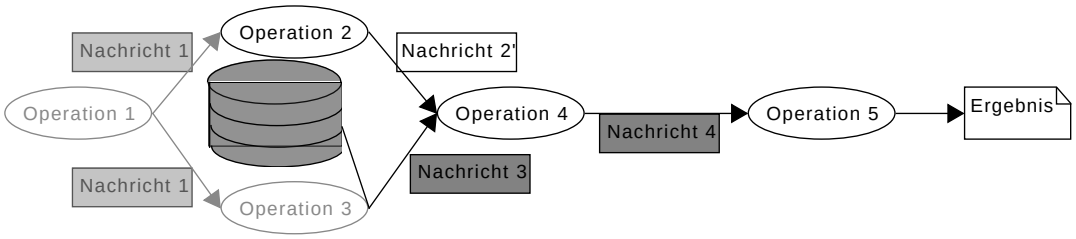

Abbildung 4.4: Beispiel Mashup Plan mit veraltetem Zwischenergebnis

Im Beispiel in Abbildung 4.4 wurden die Operationen 1 bis 3 bereits ausgeführt. Operation 4 wurde wieder umkonfiguriert. Außerdem hat sich das Ergebnis von Operation 2 verändert, da sich die verwendeten Daten verändert haben. Im Gegensatz zur Ausführung in Abbildung 4.3, muss jetzt auch Operation 2 neu ausgeführt werden.

4.4 Automatische Aktualisierung veralteter Daten

In 4.3 Erkennen veralteter Daten wird erklärt, warum nicht immer Daten der vorherigen Ausführung weiterverwendet werden können. Allerdings gibt es auch Datenquellen, die sich sehr häufig ändern.

Durch das teilweise Ausführen von Mashup Plans soll auch die Usability verbessert werden. Ändert sich jedoch eine Datenquelle am Anfang des Mashup Plan sehr häufig, so muss doch bei jeder Ausführung fast der komplette Mashup Plan ausgeführt werden. Dies führt wieder zu langen Wartezeiten für den Nutzer.

Daher soll FlexMash sich häufig ändernde Datenquellen automatisch im Hintergrund aktualisieren, und den Mashup Plan neu ausführen, sodass der User immer in kurzer Zeit aktuelle Ergebnisse erhält.

4.5 Abfragen von Zwischenergebnissen

In 4.2 Zwischenergebnisse weiterverarbeiten wurde erklärt, warum das Vorhalten von Zwischenergebnissen notwendig ist. Von diesen Zwischenergebnissen soll auch der Anwender profitieren, und diese einsehen können. Dadurch wird eine inkrementelle Entwicklung des Mashup Plans möglich, was das Modellieren vereinfacht. [RI06]

Der FlexMash Server soll erweitert werden, sodass die GUI die Ergebnisse jeder Operation abfragen kann.

4.6 Live Ausführung

In 4.1 Teilausführung von Mashup Plans wird dargelegt, dass der FlexMash Server Mashup Plans teilweise ausgeführen können soll. Dies soll auch möglich sein, wenn der Mashup Plan noch nicht vollständig modelliert ist. Dadurch kann die FlexMash GUI Mashup Plans, die der User derzeit modelliert an den FlexMash Server senden. Damit können die Wartezeiten weiter reduziert werden und die Usability weiter erhöht werden. Außerdem unterstützt dies auch die inkrementelle Entwicklung von Mashup Plans.

Der FlexMash Server unterstützt derzeit nur die Ausführung von kompletten Mashup Plans. Dies soll geändert werden, sodass auch nicht fertig modellierte Mashup Plans ausgeführt werden können.

4.7 Fortschrittsanzeige

Mashup Pläne werden derzeit synchron ausgeführt. Gibt der User den Befehl zum Ausführen eines Mashup Plans, so erhält er kein Feedback, bis der komplette Mashup Plan ausgeführt ist.

Der FlexMash Server soll so angepasst werden, dass die FlexMash GUI abfragen kann, welche Operationen eines Mashup Plans bereits ausgeführt wurden. Außerdem soll es möglich sein, dass Operationen Statusinformationen an den FlexMash Server übermitteln.

5 Konzepte

5.1 Teilausführung von Mashup Plans

Um die Teilausführung von Mashup Plans zu realisieren, muss zunächst festgestellt werden, welche Teile des Mashup Plans neu ausgeführt werden müssen. Hierfür ist es notwendig die Änderungen im Mashup Plan zu erkennen.

5.1.1 Änderungen erkennen

Bei Mashup Plans sind verschiedene Änderungen möglich:

1. Hinzufügen von Operationen
2. Entfernen von Operationen
3. Änderung der Konfiguration von Operationen
4. Änderungen im Kontroll- beziehungsweise Datenfluss

Außerdem sind beliebige Kombinationen von obigen Änderungen möglich.

Das Erkennen von hinzufügten Operationen ist trivial. Die Operationen im aktuellen Mashup Plan werden mit den Operationen im vorherigen Mashup Plan verglichen. Operationen, die im aktuellen Mashup Plan vorhanden sind, aber im vorherigen nicht, wurden neu hinzugefügt.

Das Erkennen von entfernten Operationen ist ähnlich. Hierbei findet lediglich der Vergleich umgekehrt statt: Operationen, die im vorherigen Mashup Plan enthalten waren und im aktuellen nicht, wurden entfernt.

Um Änderungen an der Konfiguration von Mashup Plans zu erkennen, müssen für jede Operation, die sowohl im aktuellen als auch im vorherigen Mashup Plan enthalten ist, alle Parameter verglichen werden. Weichen Parameter ab, so wurde die Konfiguration des Mashup Plans geändert.

Um Änderungen im Kontroll- beziehungsweise Datenfluss zu ermitteln, müssen die eingehenden und ausgehenden Verbindungen von jeder nicht neu hinzugefügten Operation mit dem vorherigen Stand verglichen werden. Kommen Kanten hinzu oder fallen Kanten weg, so hat sich der Kontrollfluss geändert.

Alle Erkennungen von Änderungen basieren darauf den aktuellen Stand des Mashup Plans mit dem vorherigen Stand zu vergleichen. Da der FlexMash Server Mashup Plans derzeit jedoch nicht speichert ist es notwendig ein Konzept für die Speicherung von Mashup Plans zu entwickeln.

5.1.2 Speichern von Mashup Plans

Da Änderungen bei Operationen, bei der Konfiguration von Operationen und im Kontroll- beziehungsweise Datenfluss auftreten können, muss alles davon gespeichert werden.

Hierfür ist es notwendig, dass die GUI einen eindeutigen Bezeichner für den Mashup Plan übermittelt. Außerdem ist es notwendig, dass die GUI einen eindeutigen Bezeichner für jede Operation übermittelt.

Für jede Operation werden dann der eindeutige Bezeichner, der Typ der durchzuführende Operation und der zugehörige Plan gespeichert.

Die Konfigurationen der Operationen können als Key-Value Pair gespeichert werden.

Der Daten- beziehungsweise Kontrollfluss kann gespeichert werden, indem man jede Kante speichert. Hierbei wird eine Kante durch den eindeutigen Bezeichner der Ausgangsoperation und der Eingangsoperation beschrieben.

Bei jedem Übermitteln des Mashup Plans von der GUI an den FlexMash Server, muss der FlexMash Server den aktuellen Plan mit dem bisherigen vergleichen und alle Änderungen speichern.

5.1.3 Auszuführenden Mashup Plan Ausschnitt erkennen

Hat sich der Mashup Plan nicht geändert, muss kein Ausschnitt des Plans neu ausgeführt werden.

Ansonsten muss man vom Start des Plans aus eine Tiefensuche auf dem zugrunde liegenden Graphen durchführen. Entweder ab der ersten geänderten, neuen oder entfernten Operation, oder ab dem ersten geänderte Daten-/ Kontrollflusseingang ändern sich die Ergebnisse. Somit muss der Mashup Plan ab der geänderten, neuen oder entfernten Operation, oder ab der Operation mit geändertem Daten- beziehungsweise Kontrollflusseingang neu ausgeführt werden.

Anschließend muss ein BPEL Workflow erstellt werden, der die auszuführenden Operationen umfasst. Dieser wird dann auf der BPEL Engine deployed und ausgeführt.

Hierbei ist es allerdings möglich, dass noch nicht alle Eingangsdaten zur Verfügung stehen. Dies kommt vor, wenn die vorherigen Operationen noch in Ausführung sind. Diese erneut auszuführen kann eine unnötige Belastung von Ressourcen sein, da die Operationen dann mehrfach ausgeführt würden. Daher muss in diesem Fall die Ausführung verzögert werden, bis die Eingangsdaten zur Verfügung stehen.

Es gibt hierbei zwei Fälle:

- Der direkte Vorgängeroperation wird gerade ausgeführt
- Der direkte Vorgängeroperation wurde verzögert, da eine seiner Vorgängeroperationen ausgeführt wird

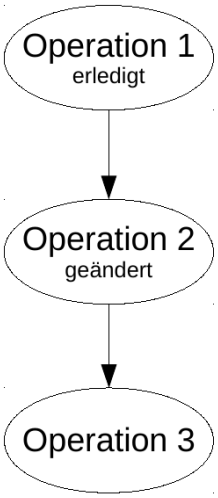

Abbildung 5.1: Sofortige Ausführung des Mashup Plan Ausschnitts nach Änderung

Wird der Vorgänger ausgeführt, so wird die Ausführung des Workflows verzögert, bis die Daten der vorherigen Operation zur Verfügung stehen.

Wird der Vorgänger ebenfalls verzögert, so muss er in den Workflow integriert werden, um möglichst wenig Overhead durch Deployoperationen zu erzeugen.

In Abbildung 5.1 wurde Operation 2 geändert. Die Ergebnisse von Operation 1 stehen zur Verfügung. Ein Workflow mit Operationen 2 und 3 kann also sofort erstellt, deployed und ausgeführt werden.

In Abbildung 5.2 wird der erste Fall skizziert. Operation 2 wurde geändert. Die Operationen 2 und 3 sollen also ausgeführt werden, allerdings ist Operation 1 noch in Ausführung. Die Ausführung von Operation 2 und 3 muss also verzögert werden, bis die Ergebnisse von Operation 1 zur Verfügung stehen.

In Abbildung 5.3 wird der zweite Fall skizziert. Operation 3 wurde geändert. Operation 2 ist bereits verzögert, da Operation 1 derzeit noch ausgeführt wird. Wenn Operation 1 abgeschlossen ist, muss ein Workflow aus den Operationen 2 und 3 erstellt, deployed und ausgeführt werden.

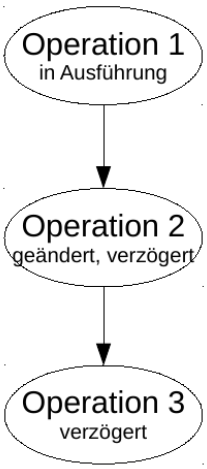

Abbildung 5.2: Verzögerte Ausführung des geänderten Mashup Plan Ausschnitts

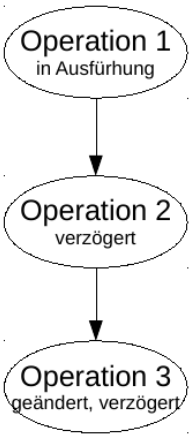

Abbildung 5.3: Verzögerte Ausführung des geänderten Mashup Plan Ausschnitts mit Vorgänger Operation

5.2 Zwischenergebnisse weiterverarbeiten

In Kapitel 3.3 wurde die erneute Ausführung von Workflowteilen in Model-as-you-go [SK11; SK12; SK13] vorgestellt. Das Vorgehen aus Model-as-you-go [SK11; SK12; SK13] ist nur möglich, wenn sich nur Konfigurationen von Operationen geändert haben. Der Workflow an sich muss unverändert bleiben.

Für die erneute Ausführung mit Zurücksetzen müsste jede Operation außerdem auch einen Compensation Handler definieren, um den Workflowsnapshot auf die geänderte Operation zurückzusetzen. Hiermit würden zusätzliche Anforderungen an alle Services gestellt werden, die in einem Mashup Plan verwendet werden sollen. Insbesondere bereits bestehende Services müssten hierfür angepasst werden.

Weiterhin entsteht durch das Zurücksetzen einer Workflow Instanz eine sehr große Abhängigkeit von der Workflow Engine. Bisher nutzt FlexMash lediglich wenige Schnittstellen der Workflowe Engine Apache ODE und hat somit keine große Abhängigkeit von der Engine, was den Austausch der Workflow Engine erleichtert.

Das Verfahren aus Model-as-you-go [SK11; SK12; SK13] ist nur in einem ausgewählten Teil der Fälle möglich. Außerdem hat es weitere Nachteile. Daher wird dieses Verfahren nicht verwendet.

In Kapitel 3.2 wurden außerdem smart re-runs in Kepler [ABJF06; Lud+06] vorgestellt. Hierbei wird theoretisch der gesamte Workflow erneut ausgeführt. Operationen, deren Daten allerdings aus den Logs der Workflow Engine bezogen werden können, werden übersprungen und stattdessen werden die Daten von einer neu eingeführten Rolle zur Verfügung gestellt. Die Daten selbst werden aus dem Log der Workflow Engine bezogen. [ABJF06]

Das Beziehen der Daten aus der Workflow Engine würde wieder eine große Abhängigkeit von der Workflow Engine bedeuten. Diese ist nicht gewünscht. Außerdem lässt auch dieses Verfahren keine Änderungen über die Konfiguration der Operationen hinaus zu.

Beide Verfahren können nicht direkt übernommen werden. Es muss also zumindest teilweise ein eigenes Konzept verwendet werden.

Nach jeder Operation werden die zurückgegebenen Daten, die an die nächste Operation weitergegeben werden zusätzlich an den FlexMash Server gesendet. Hierfür bietet der FlexMash Server einen eigenen Webservice an, der Ergebnisse von Operationen entgegen nimmt und speichert. Da das weitere Fortschreiten des Workflows nicht verzögert werden soll, findet diese Weiterleitung asynchron statt.

In Abbildung 5.4 wird dies dargestsellt. Die Operationen 1 bis 4 werden ausgeführt. Dabei werden die Ausgabedaten jeweils an die nachfolgende Operation und auch an den FlexMash Server weitergegeben.

Der FlexMash Server speichert diese Zwischenergebnisse und kann sie bei einer späteren Ausführung einer nachfolgenden Operation als Eingabedaten an die Workflow Engine weitergeben. Dies wird in Abbildung 5.5 dargestellt. Nur die Operationen 3 und 4 werden ausgeführt. Die Operation 3 erhält ihre Eingabedaten dabei vom Flexmash Server, statt von Operation 2.

5.3 Erkennen veralteter Daten

In 4.3 Erkennen veralteter Daten wird ausgeführt, dass veraltete Daten erkannt werden müssen. Veraltete Daten entstehen immer an Datenquellen Operationen, da diese Daten laden.

Eingabedaten aus vorherigen Ausführen müssen also immer zur Datenquelle zurückverfolgt werden. Sind die gecachten Daten der Datenquelle veraltet, so muss der Mashup Plan ab hier neu ausgeführt werden. Anders, als in 5.1.3 beschrieben, muss ein BPEL Workflow ab der veralteten Datenquelle erstellt, deployed und ausgeführt werden.

Da FlexMash unabhängig von der Domäne eingesetzt werden soll, gibt es kein geeignetes Verfahren, um veraltete Daten automatisch zu erkennen. Bei Flex-

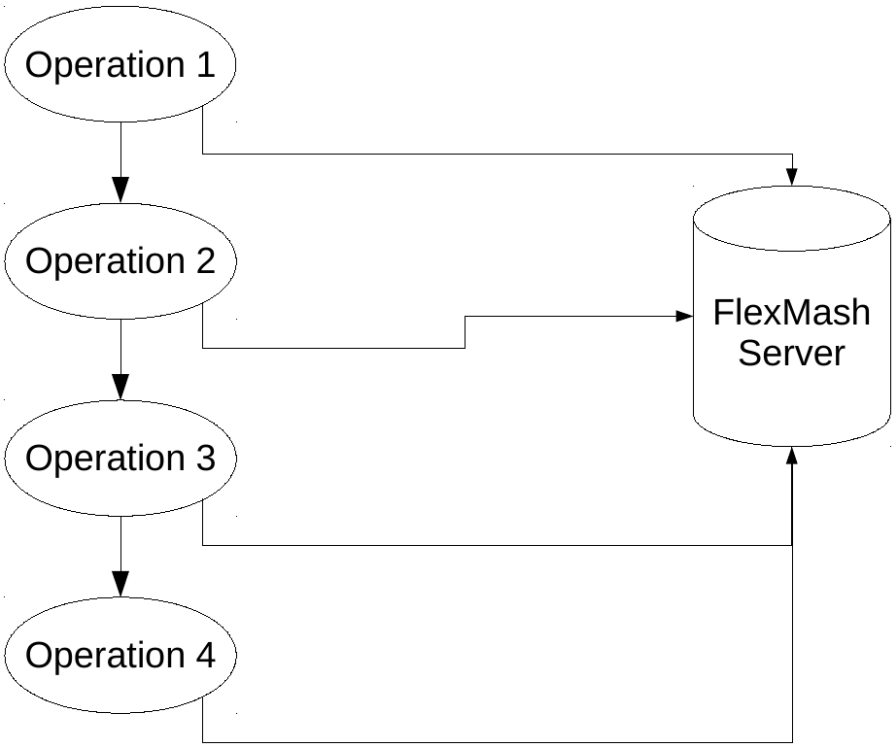

Abbildung 5.4: Veränderter Datenfluss: Ausgabedaten

Mash geht man allerdings davon aus, dass der Nutzer ein Experte der Domäne ist. Das erkennen veralteter Daten wird daher an den Experten ausgelagert.

Bei jeder Datenquellenoperationen soll ein maximales Alter der Daten konfiguriert werden können. Ist dieses überschritten, so gelten die Daten als veraltet.

Außerdem soll es möglich sein eine Ausführung des Mashup Plans ab einer bestimmten Operation zu erzwingen. So kann der Modellierer Daten manuell aktualisieren.

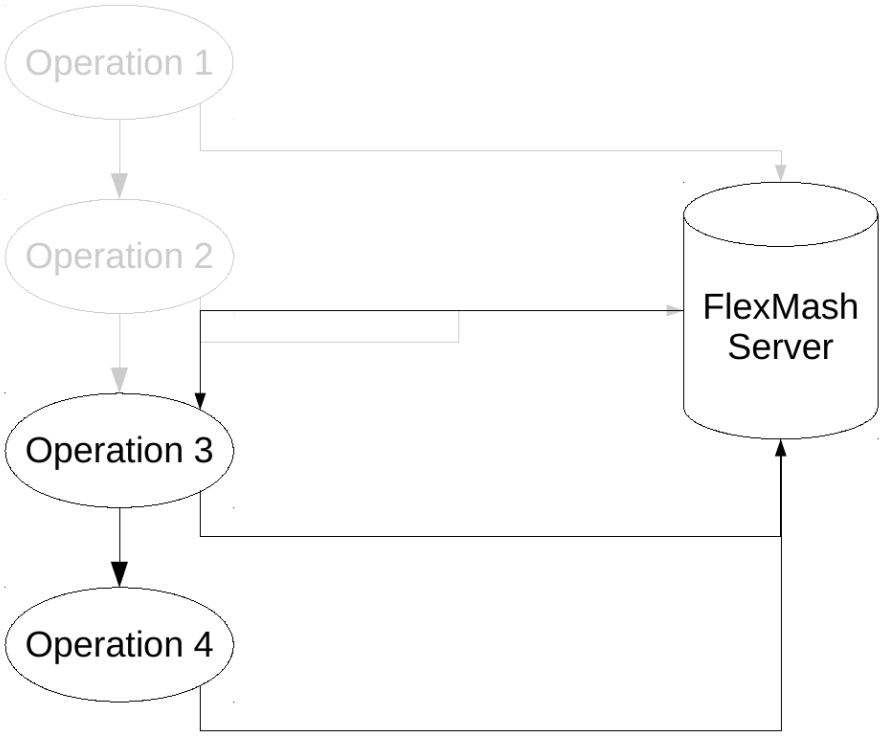

Abbildung 5.5: Veränderter Datenfluss: Eingabedaten

5.4 Automatische Aktualisierung veralteter Daten

In 4.4 Automatische Aktualisierung veralteter Daten wird gefordert, dass sich Datenquellen Operationen, die auf sich schnell ändernde Daten zugreifen sich automatisch aktualisieren.

Eine automatische Erkennung solcher Daten ist schwer möglich, da FlexMash domänenübergreifend genutzt werden können soll. Daher wird auch diese Entscheidung an den Experten übertragen. Dieser kann Datenquellen Operationen so konfigurieren, dass sie sich automatisch aktualisieren, wenn das in 5.3 vorgestellte maximale Alter überschritten ist.

Hierfür muss der FlexMash Server regelmäßig die gespeicherten Zwischenergebnisse von Datenquellen Operationen daraufhin überprüfen, ob sie ihr maximales Alter überschritten haben. Ist dies der Fall und eine automatische Aktualisierung ist vorgesehen, so wird der Mashup Plan ab der veralteten Datenquellen Operation ausgeführt.

5.5 Abfrage von Zwischenergebnissen

Die Zwischenergebnisse stehen dem FlexMash Server durch die in 5.2 vorgestellten Konzepte zur Verfügung. Für den Zugriff auf diese Daten muss also nur eine Schnittstelle zur Verfügung gestellt werden.

Da jederzeit Zwischenergebnisse abgefragt werden können und nach Abschnitt 5.4 Daten automatisch aktualisiert werden können, können auch Echtzeitdaten sehr gut verarbeitet werden und Live Mashups erzeugt werden.

5.6 Live Ausführung

Für die Live Ausführung ist größtenteils die FlexMash GUI verantwortlich. Der FlexMash Server unterstützt durch die vorgestellten Konzepte aus 5.1 und 5.2 die teilweise Ausführung von Mashup Plans. Die FlexMash GUI kann immer, wenn eine Operation alle Eingänge, sowie alle Parameter definiert hat den Mashup Plan an den Server senden und somit ausführen. Durch die Schnittstelle aus 5.5 können Zwischenergebnisse jederzeit abgefragt werden.

Durch häufige Aufrufe der FlexMash GUI an den FlexMash Server können hierbei viele offene Requests entstehen. Damit die FlexMash GUI nicht zu viele offene Requests an den FlexMash Server hat, muss dieser Mashup Plans anders als bisher asynchron ausführen. Dadurch kann auch nicht mehr das Ergebnis des kompletten Mashup Plans zurückgeliefert werden. Dieses muss über eine eigene Schnittstelle zur Verfügung gestellt werden.

5.7 Fortschrittsanzeige

Durch den in 5.2 vorgestellten, veränderten Datenfluss wird der FlexMash Server informiert, wenn eine Operation ausgeführt ist. Dies dient als Erkennung, dass eine Operation ausgeführt wurde.

Zusätzlich stellt der FlexMash Server eine Schnittstelle zur Verfügung, über die Operationen einen Prozentwert an den FlexMash Server übermitteln können. Hierfür benötigt die Operation ihren eindeutigen Bezeichner als zusätzlichen Parameter. Wie und ob diese erweiterte Fortschrittsanzeige genutzt wird, obliegt demjenigen, der eine Operation implementiert.

Der FlexMash Server stellt außerdem eine Schnittstelle zur Verfügung, über die die FlexMash GUI den Fortschritt einer Operation abfragen kann.

6 Implementierung

In diesem Kapitel wird die Implementierung der Konzepte aus Kapitel 5 Konzepte in FlexMash vorgestellt.

6.1 Datenmodell

In diesem Abschnitt wird das Datenmodell vorgestellt. Das vollständige Datenmodell ist in Abbildung 6.1 dargestellt.

6.1.1 Modell des Mashup Plans

In 5.1.2 Speichern von Mashup Plans wurde ausgeführt, dass Mashsup Plans vom FlexMash Server gespeichert werden müssen. Hierbei müssen alle Operationen, hier Knoten (englisch Nodes) als Knoten des Graphs gespeichert werden. Hierfür werden die Attribute id, plan, name und dbId nötig gespeichert.

- **id** ist der eindeutige Bezeichner, der von der GUI generiert wird
- **plan** ist der eindeutige Bezeichner, des Mashup Plans zu dem der Knoten gehört
- **name** gibt die Operation an, die vom Knoten durchgeführt wird
- **dbId** gibt ebenfalls die Operation an, die vom Knoten durchgeführt wird und soll zukünftig das Attribut name als Identifikator der Operation ablösen

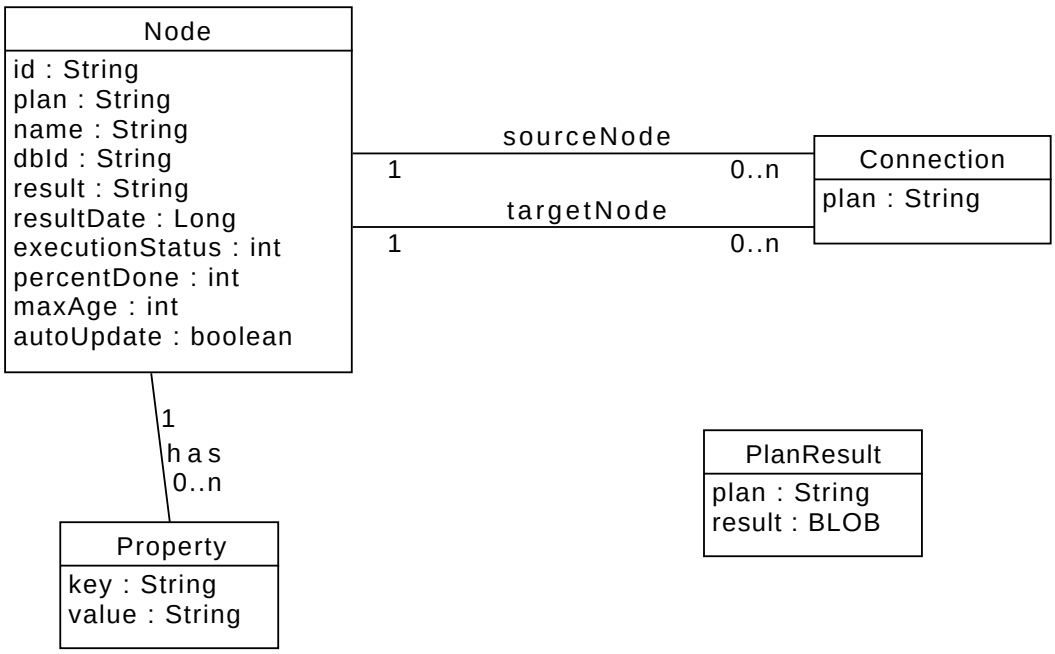

Abbildung 6.1: Datenmodell eines Mashup Plans

Für jeden Knoten müssen auch die Eigenschaften gespeichert werden. Hierfür dient das Datenmodell Property (englisch für Eigenschaft). Jede Eigenschaft ist ein Key-Value-Pair.

Außerdem müssen für jeden Knoten die eingehenden und ausgehenden Verbindungen (englisch Connections) zu anderen Knoten gespeichert werden. Diese Verbindungen repräsentieren den Daten- beziehungsweise Kontrollfluss.

Jede Verbindung hat das Attribut plan, sourceNode und targetNode.

- **plan** ordnet die Verbindung einem Mashup Plan zu
- **sourceNode** ist die id des Knotens von dem die Verbindung ausgeht
- **targetNode** ist die id des Knotens zu dem die Verbindung führt

6.1.2 Speichern des Ausführzustands

Um den Auszuführenden Ausschnitt des Mashup Plans zu erkennen, ist es notwendig zu speichern, welche Operationen gerade ausgeführt werden. Außerdem müssen die Knoten bekannt sein, welche gerade für die Ausführung vorgehalten, beziehungsweise verzögert werden.

Hierfür dient das Attribut executionStatus eines Knotens.

- 0: noch nicht ausgeführt
- 1: Ausführung zurückgestellt
- 2: in Ausführung
- 3: Ausführung beendet

6.1.3 Zwischenergebnisse speichern

In Kapitel 5.2 Zwischenergebnisse weiterverarbeiten wurde ausgeführt, dass für jede Operation das Ergebnis der letzten Ausführung gespeichert werden muss. Hierfür besitzt jeder Knoten das Attribut result (deutsch: Ergebnis). Da alle derzeitigen Operationen in FlexMash JSON als Austauschformat nutzen, ist das result Attribut in der prototypischen Implementierung als String definiert.

6.1.4 Erkennen veralteter Daten

Das Erkennen veralteter Daten erfolgt, wie in Kapitel 5.3 ausgeführt, auf Basis eines maximalen Alters für Ergebnisse. Dieses maximale Alter wird im Attribut maxAge eines Knotens gespeichert.

- -1 bedeutet dabei, dass die Daten kein maximales Alter besitzen
- 0 bedeutet hierbei, dass es sich bei der Operation nicht um eine Datenquellen Operation handelt
- alle anderen Werte geben das maximale Ergebnisalter in Sekunden an

Um zu überprüfen, wann das maximale Alter erreicht ist, muss außerdem gespeichert werden, wann das aktuelle Ergebnis erzeugt wurde. Hierfür dient das Attribut resultDate (deutsch: Datum des Ergebnisses) eines Knotens.

6.1.5 Automatische Aktualisierung veralteter Daten

Es muss gespeichert werden, welche Datenquellen Operationen automatisch ihre Ergebnisse aktualisieren sollen. Hierfür besitzt jeder Knoten das Attribut autoUpdate. Dieses Attribut wird nur beachtet, wenn der Knoten über den maxAge Wert als Datenquellen Operation definiert wird. autoUpdate ist ein boolean – die Bedeutung der Belegung entsprechend trivial.

6.1.6 Fortschrittsanzeige

Für die Fortschrittsanzeige wird zum einen das Attribut executionStatus (siehe 6.1.2) verwendet. Außerdem wurde im Kapitel 5.7 vorgestellt, dass Operationen eine prozentuale Fertigstellung übermitteln können. Diese wird im Attribut percentDone (deutsch: Prozent erledigt) eines Knotens gespeichert.

6.1.7 Zugriff auf Ergebnis des Mashup Plans

Das Ergebnis des Mashup Plans wird als eigenes Objekt gespeichert. Es besitzt als Attribute plan und result. Das Attribut plan ist der eindeutige Bezeichner des Mashup Plans. Das Attribut result ist das Ergebnis als Datei.

6.2 Architektur

In diesem Kapitel wird die geänderte Architektur des FlexMash Servers vorgestellt. Eine Übersicht über die Architektur ist in Abbildung 6.2 abgebildet.

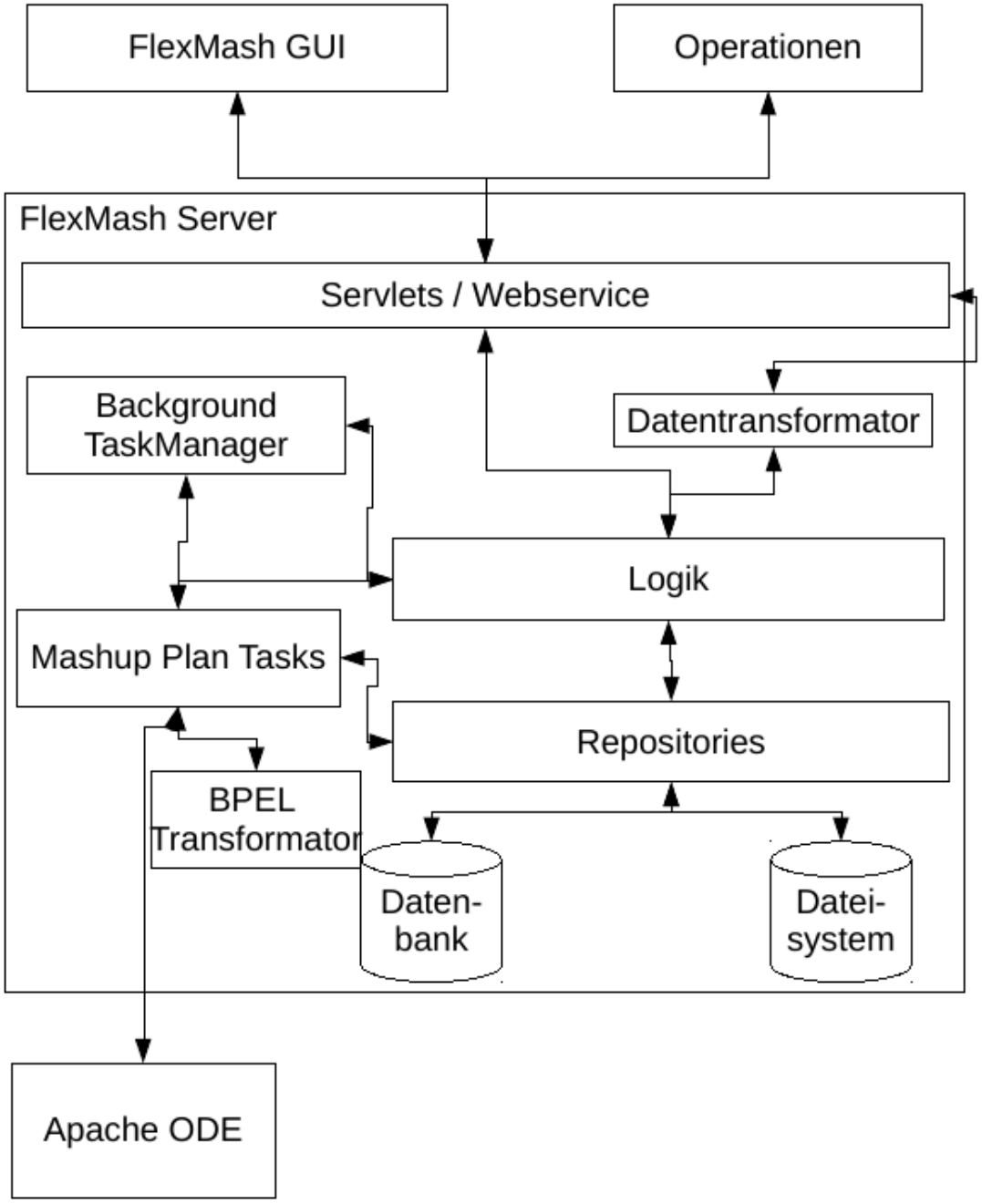

Abbildung 6.2: Neue FlexMash Architektur

6.2.1 Datenhaltung

Die im Datenmodell vorgestellten Daten werden zum Teil in einer MySQL Datenbank und zum Teil im Dateisystem gespeichert. Nodes, Connections und Properties werden in der MySQL Datenbank gespeichert, da so ein einfacher, strukturierter Zugriff möglich ist.

Die Ergebnisse von Mashup Plans werden im Dateisystem gespeichert. Dies hat den Grund, dass die Ergebnisse jedes beliebige Format haben können.

Der Zugriff auf die gespeicherten Objekte erfolgt über Repositories gemäß dem Repository Pattern. [Rep] Dadurch ist ein einfacher Austausch der Speicherarten gewährleistet.

6.2.2 Datentransformator

Da der FlexMash Server Mashup Plans speichert, verwendet er jetzt ein eigenes Datenmodell (Kapitel 6.1), um Mashup Plans zu speichern. Der Datentransformator übernimmt die Transformation vom JSON der GUI in das Server eigene Datenmodell.

6.2.3 Auszuführenden Mashup Plan Ausschnitt erkennen

Die Logikkomponente in der Architekturübersicht (Abbildung 6.2) berechnet welche Teile des Mashup Plans ausgeführt werden müssen und speichert den geänderten Mashup Plan. Außerdem entscheidet sie, welche Art von Mashup Plan Task generiert werden. Dabei wird zwischen sofort auszuführenden und für die Ausführung vorgemerkten Mashup Plans unterschieden. Wurde der auszuführende Teilplan generiert, so wird er vom BackgroundTaskManager ausgeführt. Dieser verwaltet jeden Mashup Plan Task in einem eigenen Thread. Sobald der Task generiert ist, kann eine Antwort an die GUI gesendet werden, da die weitere Ausführung asynchron abläuft.

6.2.4 Zwischenergebnisse weiterverarbeiten

Wird ein Mashup Plan Task ausgeführt, so übernimmt ein angepasster BPEL Transformator die Generierung des BPEL Workflows. Hierbei wird nach jeder Operation ein asynchroner Aufruf an den FlexMash Server zum Übermitteln der Ergebnisse hinzugefügt.

Außerdem wird bei der ersten Operation gegebenenfalls die Rolle ausgetauscht, über die die Eingabedaten zur Verfügung gestellt werden. Werden die Eingabedaten vom FlexMash Server zur Verfügung gestellt, so übergibt dieser die Eingabedaten per SOAP Nachricht an die Workflow Engine.

6.2.5 Automatische Aktualisierung veralteter Daten

Für die automatische Aktualisierung veralteter Daten führt der Background-TaskManager permanent einen Task aus, der veraltete Daten sucht. Findet dieser veraltete Daten, so generiert er einen Mashup Plan Task, der den zugehörigen Mashup Plan ab dem veralteten Datensatz ausführt.

6.3 Schnittstellen

Der FlexMash Server besitzt außerdem diverse Schnittstellen, die angepasst oder neu hinzugefügt wurden. Die Funktionalität dieser Schnittstellen ist selbsterklärend, da es sich um simple CRUD Operationen handelt. Die meisten Schnittstellen werden daher ohne weitere Erklärung aufgelistet.

- **Mashup Plan ausführen** ruft die in Kapitel 6.2.3 aufgeführte Aktion auf
- **Operation löschen** löscht eine Operation, einschließlich aller Properties und Connections
- **Mashup Plan löschen** löscht alle Daten eines Mashup Plans

- **Ergebnis übermitteln** speichert das übermittelte Ergebnis als Ergebnis einer Operation
- **Ausführzustand übermitteln** aktualisiert den Prozentwert, zu dem eine Operation ausgeführt ist
- **Operationsergebnis abfragen** liefert das gespeicherte Ergebnis einer Operation zurück
- **Mashup Plan Ergebnis abfragen** liefert das Endergebnis eines Mashup Plans zurück
- **Operationszustand abfragen** liefert alle Informationen über eine Operation zurück

7 Beispielszenario

Die neuen und geänderten Features von FlexMash sollen anhand eines einfachen Beispielszenarios demonstriert werden. Dieses Szenario sowie seine Implementierung wird in diesem Kapitel vorgestellt.

7.1 Szenario

Im Beispielszenario möchte ein Marketing Manager die Auswirkungen von Werbekampagnen auf die Verkaufszahlen untersuchen. Hierfür möchte er ein Diagramm generieren, aus dem er ablesen kann, welche Werbekampagnen in welchen Zeiträumen geschaltet waren. Außerdem sollen aus dem Diagramm die Verkaufszahlen über die verschiedenen Vertriebswege abgelesen werden können.

Die Datenquellen sind somit die Verkaufszahlen, sowie die durchgeführten Werbekampagnen.

7.2 Mashup Plan

Die Datenquellen und Operationen, die für das Szenario benötigt werden, werden in diesem Abschnitt vorgestellt.

Der komplette Mashup Plan des Beispielszenarios ist in Abbildung 7.1 dargestellt.

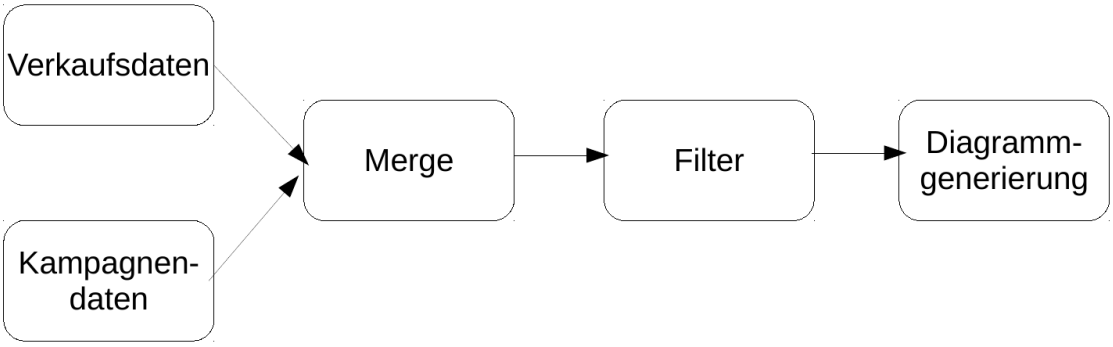

Abbildung 7.1: Mashup Plan des Beispielszenarios

Datenquellen

Verkaufszahlen

Die Firma vertreibt ihr Produkt auf zwei Wegen. Zum einen wird das Produkt über einen Webshop verkauft. Außerdem kann das Produkt in Einkaufsläden erworben werden. Für jeden Vertriebsweg wird die Summe der verkauften Artikel pro Tag gespeichert.

Werbekampagnen

Die Firma verwendet drei verschiedene Werbemedien. Es wird Radiowerbung, Fernsehwerbung und Onlinewerbung geschaltet. Jede Werbekampagne hat dabei ein Startdatum, sowie ein Enddatum. Außerdem können auch mehrere Kampagnen gleichzeitig aktiv sein.

Operationen

Datenquellen Operationen

Es werden für beide Datenquellen Operationen benötigt, die die Datenquellen auslesen.

Merge

Die beiden Datenquellen müssen zusammengeführt werden, sodass für jeden Tag die Information vorliegt, wie viele Produkte via Webshop verkauft wurden, wie viele Produkte in Läden verkauft wurden, sowie welche Werbekampagnen stattgefunden haben.

Filter

Die Daten sollen in einem bestimmten Zeitraum betrachtet werden. Da die Datenquellen jedoch auch Daten enthalten können, die außerhalb des Zeitraums liegen, müssen die relevanten Daten herausgesucht werden. Hierfür wird ein Filter benötigt, über den das Start- und Enddatum konfiguriert werden kann. Alle Daten außerhalb des konfigurierten Zeitraums werden vom Filter aus dem zusammengeführten Datensatz entfernt.

Diagrammgenerierung

Bei der Diagrammgenerierung werden die gefilterten Daten für die Ausgabe weiterverarbeitet. Ein Beispieldiagramm wird in Abbildung 7.2 gezeigt.

Im Diagramm wird die Anzahl der im Webshop verkauften Artikel als schwarze Linie dargestellt, die Anzahl der in Läden verkauften Artikel als weiße Linie. Der Hintergrund stellt die stattfindenden Werbekampagnen dar.

- grau = keine Werbekampagne
- türkis = Onlinewerbung

- grün = Fernsehwerbung
- helles rot = Radiowerbung
- lila = Onlinewerbung und Fernsehwerbung
- beige = Onlinewerbung und Radiowerbung
- rot = Fernsehwerbung und Radiowerbung
- blau = Onlinewerbung, Fernsehwerbung und Radiowerbung

7.3 Implementierung

7.3.1 Testdaten Generator

Um Datensätze für die Ausführung des Mashup Plans zu haben wurde ein Testdatengenerator implementiert. Dieser ist ein Java Konsolenprogramm. Er generiert für eine konfigurierbare Anzahl an Tagen, rückwärts vom aktuellen Datum, Datensätze. Diese können entweder ausgegeben oder als SQL Dateien gespeichert werden.

Die Verkaufsdatensätze umfassen jeweils das Datum, die Verkaufsart, sowie die Anzahl. Pro Tag werden also maximal zwei Datensätze generiert: ein Datensatz für die Webshop Verkäufe und ein Datensatz für die Ladenverkäufe. Die Anzahl der verkauften Artikel liegt zwischen einem und 100 Artikeln, bei 0 Artikeln wird kein Datensatz erzeugt. Die Anzahl der verkauften Artikel wird von einem Zufallsgenerator erzeugt.

Die Datensätze der Werbekampagnen bestehen jeweils aus einem Startdatum, einem Enddatum und der Kampagnenart. Eine Werbekampagne hat eine maximale Dauer von 7 Tagen. Die Kampagnen werden ebenfalls per Zufallsgenerator erzeugt.

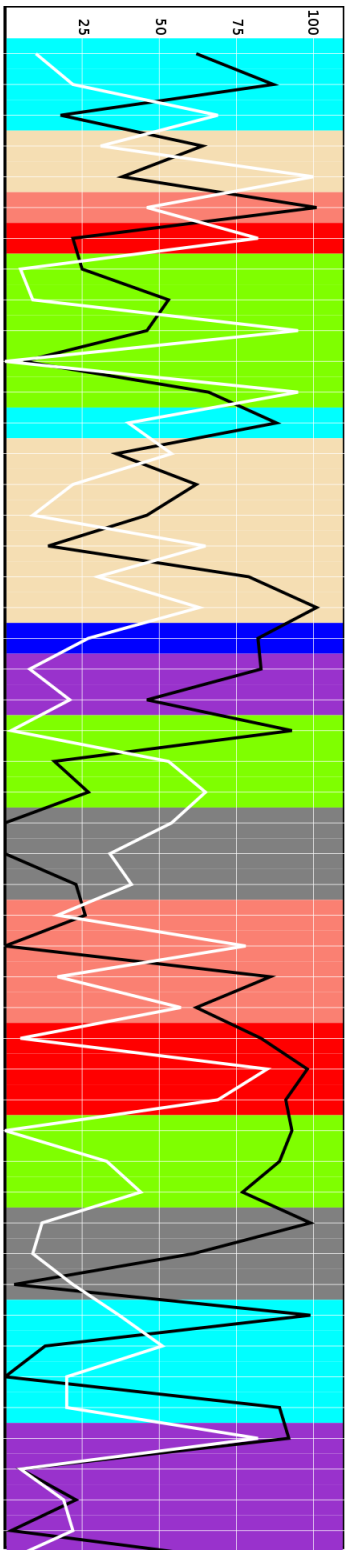

Abbildung 7.2: Ergebnisdiagramm des Beispielszenarios

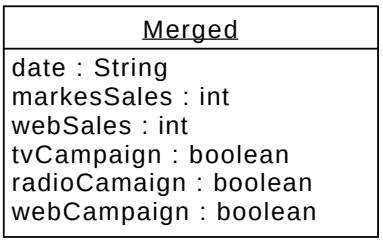

Abbildung 7.3: UML Darstellung eines zusammengeführten Datensatzes im Beispielszenario

7.3.2 Datenquellen Operationen

Da es sich um eine einfache Beispielanwendung handelt, werden als Datenquellen jeweils SQL Datenbanken verwendet. Für diese bietet FlexMash bereits eine Datenquellen Operation, den SQLExtractor. Hierdurch kann außerdem gezeigt werden, dass die Konzepte für die Teilausführung (Kapitel 5.1) und die Weiterverarbeitung von Zwischenergebnissen (Kapitel 5.2) auch auf bereits implementierte Operationen angewendet werden können.

7.3.3 Merge

Die Mergeoperation verarbeitet die beiden JSON Arrays der SQLExtractors und führt diese zusammen. Hierbei wird für jeden Tag, zwischen dem frühesten und spätesten Datum innerhalb der beiden Datenquellen ein Datensatz erzeugt. Dieser Datensatz umfasst das Datum, die Anzahl der Webshop Verkäufe, die Anzahl der Ladenverkäufe und die stattfindenden Werbekampagnen. Eine UML Darstellung ist in Abbildung 7.3 dargestellt.

Die Datensätze werden als JSON Array ausgegeben.

Die Mergeoperation unterstützt die Fortschrittsanzeige (Kapitel 5.7) und demonstriert damit deren Funktionalität.

7.3.4 Filter

Die Filteroperation iteriert über das JSON Array der Mergeoperation und entfernt alle Datensätze, die nicht im konfigurierten Zeitraum liegen.

Die verbleibenden Datensätze werden ebenfalls als JSON Array weitergegeben.

Auch die Filteroperation unterstützt die Fortschrittsanzeige.

7.3.5 Diagrammgenerator

Der Diagrammgenerator verarbeitet das JSON Array der Filteroperation und generiert daraus ein SVG Bild (Abbildung 7.2).

8 Zusammenfassung

In dieser Arbeit wurde das Data Mashup Tool FlexMash verbessert.

Das Hauptziel war, dass Mashup Plans teilweise ausgeführt werden können. Dadurch soll die Usability und Performance von FlexMash verbessert werden.

Durch die in dieser Arbeit erarbeiteten und implementierten Konzepte, werden bei jeder Ausführung eines Mashup Plans nur die minimale Anzahl an auszuführenden Operationen ausgeführt. Dies ist möglich, da der FlexMash Server jetzt die Ergebnisse jeder Operation als Zwischenergebnisse eines Mashup Plans speichert und die Zwischenergebnisse für eine Ausführung nachfolgender Operationen zur Verfügung stellen kann. Somit kann ein Mashup Plan ab jeder beliebigen Stelle erneut ausgeführt werden.

Um die Datenqualität bei gespeicherten Zwischenergebnissen sicherzustellen, wurden Verfahren Konzeptioniert und in FlexMash implementiert, die sicherstellen, dass Datenquellenoperationen beim Ausführen eines Mashup Plans erneut ausgeführt werden, wenn die Daten nicht mehr der gewünschten Qualität entsprechen.

FlexMash unterstützt jetzt außerdem die automatische Aktualisierung von Daten aus Datenquellen Operationen. Dies kombiniert die Sicherstellung der Datenqualität mit der verkürzten Ausführungszeit, der teilweisen Ausführung von Mashup Plans. Außerdem können so Daten, die sich sehr häufig ändern in einem Mashup verarbeitet werden, ohne dass der Anwender den Mashup Plan manuell neu ausführen muss.

Außerdem wurden weitere Schnittstellen implementiert, die die durch die Änderungen vorhanden Daten nutzen, um die Usability weiter zu erhöhen.

9 Ausblick

Aufbauend auf dieser Arbeit kann FlexMash weiter verbessert werden und somit nutzerfreundlicher gestaltet werden. Weiterführende Ideen finden sich in diesem Kapitel.

9.1 Datenquellen für Caching optimieren

Derzeit muss der Benutzer bei Datenquellen manuell angeben, wie lange die Daten aktuell sind. Zum einen stellt dies zusätzlichen Aufwand für den Benutzer dar. Außerdem können Fehlkonfigurationen des Users zu unnötigen Datenaktualisierungen führen, was wiederum zu erhöhter Rechenlast führt.

Da der FlexMash Server den Zeitpunkt kennt, zu dem er die Daten von der Datenquelle bezogen hat, könnte der FlexMash Server mit der Datenquelle die Aktualität der Daten abgleichen, bevor die Daten abgefragt werden. Alternativ könnte die Datenquelle auch ein Ablaufdatum für die Daten an den FlexMash Server senden.

9.2 Workflow Deployments optimieren

Derzeit entfernt der FlexMash Server deployte BPEL Pläne nach deren Ausführung. Allerdings benötigen auch Deployment und Undeployment von BPEL Plänen in der Workflow Engine Rechenleistung und Zeit. Daher könnte der FlexMash Server speichern, welche Workflowausschnitte derzeit auf der Workflow Engine deployed sind. Workflow Ausschnitte werden erst dann entfernt,

wenn der Ausschnitt nicht mehr mit dem Mashup Plan übereinstimmt. Außerdem wird ein Workflow Ausschnitt nur dann deployed, wenn er noch nicht auf der Workflow Engine deployed ist.

9.3 Visualisierung von Zwischenergebnissen

Mit dieser Arbeit unterstützt der FlexMash Server das Abfragen von Zwischenergebnissen. Die Ergebnisse jeder Operation können damit eingesehen werden. Allerdings speichert der FlexMash Server die Daten nur im Austauschformat zwischen den Operationen. Dieses Austauschformat ist nicht benutzerfreundlich. FlexMash könnte erweitert werden, sodass die Zwischenergebnisse für den Domänenexperten verständlich visualisiert werden.

Literaturverzeichnis

[ABJF06] I. Altintas, O. Barney, E. Jaeger-Frank. „Provenance collection support in the kepler scientific workflow system“. In: *International Provenance and Annotation Workshop*. Springer. 2006, S. 118–132 (zitiert auf S. 26, 27, 40).

[ANP12] S. Aghaee, M. Nowak, C. Pautasso. „Reusable decision space for mashup tool design“. In: *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM. 2012, S. 211–220 (zitiert auf S. 20).

[AP13] S. Aghaee, C. Pautasso. „Live mashup tools: challenges and opportunities“. In: *Proceedings of the 1st International Workshop on Live Programming*. IEEE Press. 2013, S. 1–4 (zitiert auf S. 21).

[Cas11] F. Casati. „How end-user development will save composition technologies from their continuing failures“. In: *International Symposium on End User Development*. Springer. 2011, S. 4–6 (zitiert auf S. 21).

[DM14] F. Daniel, M. Matera. *Mashups: Concepts, Models and Architectures*. Springer, 2014 (zitiert auf S. 9, 13–20).

[HM16] P. Hirmer, B. Mitschang. „FlexMash–Flexible Data Mashups Based on Pattern-Based Model Transformation“. In: *Rapid Mashup Development Tools*. Springer, 2016, S. 12–30 (zitiert auf S. 9, 25).

[Hir+15] P. Hirmer, P. Reimann, M. Wieland, B. Mitschang. „Extended Techniques for Flexible Modeling and Execution of Data Mashups.“ In: *DATA*. 2015, S. 111–122 (zitiert auf S. 9, 11, 25, 26).

[Lud+06] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao. „Scientific workflow management and the Kepler system“. In: *Concurrency and Computation: Practice and Experience* 18.10 (2006), S. 1039–1065 (zitiert auf S. 26, 40).

[Nai15] V. K. Naik. „A framework for business mashup applications“. In: *International Conference on Web Engineering*. Springer. 2015, S. 617–620 (zitiert auf S. 9).

[Pap08] M. Papazoglou. *Web services: principles and technology*. Pearson Education, 2008 (zitiert auf S. 21–23).

[RI06] A. Repenning, A. Ioannidou. „What makes end-user development tick? 13 design guidelines“. In: *End User Development*. Springer, 2006, S. 51–85 (zitiert auf S. 21, 33).

[Rep] URL: https://martinfowler.com/eaaCatalog/repository.html (zitiert auf S. 52).

[SK11] M. Sonntag, D. Karastoyanova. „Enforcing the repeated execution of logic in workflows“. In: (2011) (zitiert auf S. 27, 28, 40).

[SK12] M. Sonntag, D. Karastoyanova. „Ad hoc Iteration and Re-execution of Activities in Workflows“. In: *International Journal On Advances in Software* 5.1-2 (2012) (zitiert auf S. 27, 28, 40).

[SK13] M. Sonntag, D. Karastoyanova. „Model-as-you-go: an approach for an advanced infrastructure for scientific workflows“. In: *Journal of grid computing* 11.3 (2013), S. 553–583 (zitiert auf S. 27, 28, 40).

Alle URLs wurden zuletzt am 16. 01. 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift