BACHELORARBEIT

Entwicklung eines Interaktionsprotokolls für die plattformgestützte Abrechnung bei REST-basierten Service-Mashups

von

Elmar Jakobs

eingereicht am 12.05.2011 beim
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
des Karlsruher Instituts für Technologie

Referent: Prof. Dr. Stefan Tai
Betreuer: Dr. Christian Zirpins
Betreuer: Dr. Volker Kuttruff (CAS AG)

Heimatanschrift:
Untere Hart 28
67117 Limburgerhof

Studienanschrift:
Kriegsstraße 51
76133 Karlsruhe

Inhaltsverzeichnis

Einleitung

In Zeiten des Cloud Computing liegt es im Trend, über das Internet verschiedenste Applikationen oder Dienste zu komponieren und damit einen Mehrwert zu generieren. Dabei werden diese Programme abstrakt als Services aufgefasst, die über eine offene Programmierschnittstelle aufgerufen werden können. Kombiniert man nun mehrere solcher Services zu einem neuen Dienst, dann spricht man von einem Mashup. Dieses kann wiederum selbst als Service bereitgestellt oder von einem Nutzer aufgerufen werden. Dabei bringt das Mashup einen höherwertigen Nutzen mit sich, als es ihn ohne die Verknüpfung dieser Dienste gegeben hätte. [vgl. CCHZ08; S. 4-5]
Beliebt sind hierbei sogenannte RESTful Service-Mashups, die auf den Technologien des Webs basieren. Konkret bedeutet dies, dass diese Dienste auf Anwendungsebene das HTTP-Protokoll einsetzen. Im Gegensatz dazu verwenden die Web Services aus der WS*-Welt weitere Technologien wie SOAP oder auch WSDL. Die Web-Technologien dienen hierbei nur dem Transport. Die Verwendung des REST-basierten Ansatzes zeichnet sich vor allem durch die weite Verbreitung und die Einfachheit der verwendeten Technologien aus. [vgl. RiRu07; S. 299-300]
Probleme bei der Erstellung und Nutzung solcher Mashups sind unter anderem das Auffinden gesuchter Services und bei der entgeltlichen Nutzung die Protokollierung und Abrechnung der Aufrufe. Vor allem wenn es darum geht, Services oder Mashups für Geld anzubieten, wirkt sich die Einfachheit von REST nachteilig aus, da ein solcher Bezahlungsvorgang protokollseitig von HTTP nicht unterstützt wird. [vgl. FGMF99]
Um diesen Problemen zu begegnen, schlägt das Cocktail-Forschungsprojekt vom Bundesministerium für Bildung und Forschung den Aufbau einer Plattform vor, „die es ermöglicht Funktionalitäten in unterschiedlichen Granularitäten, das heißt als Dienste, Mashups oder Applikationen**,** in einer einheitlichen Form bereitzustellen und zu höherwertigen Diensten kombinierbar sowie gestuft gewerblich nutzbar zu machen“. [Cock11] Die dem zugrunde liegende Idee ist, dass diese Plattform unter anderem ein Service-Repository besitzt, in dem gesuchte Dienste leichter gefunden werden können. Außerdem werden benötigte Infrastrukturdienste zur Verfügung gestellt, die das Mitprotokollieren von Service- und Mashup-Aufrufen sowie die folgende Rechnungsstellung übernehmen.

1.1 Problemstellung

Der Fokus dieser Arbeit soll im Folgenden auf der Untersuchung des Zahlungsvorgangs bei kommerziellen Mashups liegen. Insbesondere sollen die Besonderheiten bei der Benutzung eines Prepaid-Preissystems untersucht

werden, nämlich dass bei einer Prepaid-Bezahlung der Kunde im Vorhinein die gewünschte Leistung begleichen muss. Dabei kann das Problem auftreten, dass der Kunde bezahlt, aber keine Leistung erhält.
Dieses grundsätzliche Problem wirkt sich im Mashup-Kontext ebenfalls stark aus, da hier viele Fehlerfälle die „korrekte“ Lieferung der geforderten Leistung an den Kunden verhindern können. Diese setzen sich einerseits aus anwendungsorientierten Fehlern zusammen und andererseits aus technischen Problemen, die durch die zugrundeliegenden Protokolle nicht behandelt werden. Hier wirkt sich die Einfachheit von REST negativ aus, da dadurch viele Sicherheiten bezüglich der Behandlung von Fehlern nicht gegeben werden können.
Daher ist es wichtig für das Funktionieren von kommerziellen Mashup-Anwendungen, dass auch im Fehlerfall die korrekte Abrechnung sichergestellt ist.
Die Abrechnung erfolgt im Cocktail-Kontext über das Mitprotokollieren der Service-Aufrufe, dass dann in die Prepaid-Kontenverwaltung einfließt. Insofern bezieht sich in unserem Fall die Abrechnungsproblematik auf ein korrektes Metering der Mashup- und Service-Aufrufe. Metering bezeichnet hierbei das Zählen und Protokollieren der Service-Aufrufe.
Die nachfolgende Untersuchung dieses Sachverhaltes basiert auf einigen Annahmen bezüglich des zugrundeliegenden Service- und Mashup-Modells. Zunächst soll das zugrundeliegende Service-Modell kurz erläutert werden. Das Cocktail-Projekt nimmt an, dass bei einer Anfrage an einen Dienst die entsprechende Response-Nachricht die angeforderte Lösung enthält und nicht weitere Aufrufe seitens des Service-Nutzers notwendig sind. Zum Beispiel könnte man sich einen Dienst vorstellen, der bei Aufruf CRM-Datensätze an den Nutzer zurückgibt. Hierfür wäre nur ein Request notwendig. Ein Beispiel bei dem das Ergebnis nicht durch eine einzelne Anfrage- beziehungsweise Antwortnachricht erzielt werden kann, ist der „doodle“ – Service, bei dem man jeweils einen Aufruf für das Erstellen einer Umfrage und dann für das Anlegen von Benutzern starten muss. [vgl. REST11] Trotz dieser starken Einschränkung des Modells sind viele Szenarien vorstellbar, die diesem einfachen Request/ Response-Verhalten folgen. Zu nennen sind hier zum Beispiel Dienste, die lediglich Daten aus einer Datenbank zu einem bestimmten Suchbegriff zurückgeben. Hierzu zählen nicht nur der bereits erwähnte CRM-Dienst, sondern auch beliebte Schnittstellen wie die Google-Web-Search-API, bei der zu einem Suchbegriff die entsprechenden Ergebnisse zurückgegeben werden. Auch die Problemstellung hinsichtlich der Lieferung und Bezahlung wird durch diese Einschränkung des Modells nicht trivial, sondern existiert auch weiterhin.
Eine weitere Annahme betrifft die Eingabeparameter, mit denen ein Service aufgerufen werden kann. Im Folgenden wird davon ausgegangen, dass diese Menge an Parametern einschränkbar ist, also dass ein Service auch mit einer Teilmenge der ursprünglichen Input-Parameter aufrufbar ist. Des Weiteren ist jeder Service charakterisierbar hinsichtlich der Möglichkeit ihn kosten-

frei stornieren zu können. Hierbei ist zum Beispiel denkbar, dass ein Service, der lediglich eine Berechnung durchführt und damit nur Rechenleistung verbraucht, eine solche Möglichkeit bietet. Dem gegenüber ermöglicht ein Dienst, der bei Aufruf zum Beispiel physisch Briefe verschickt und damit „erhebliche“ Kosten trägt, nur eine Stornierung bei Bezahlung der bereits erbrachten Leistung. Außerdem ist jedem Service, der in das Service-Repository der Plattform eingestellt wird, ein Preis zugeordnet. In unserem Fall ein Prepaid-Preis. Dieser kann aus einem fixen Betrag bestehen oder auch dynamisch von der zu erbringenden Leistung abhängen. Möglich ist auch ein aus beiden Komponenten zusammengesetzter Preis.
Gleiches gilt für den Preis eines Service-Mashups, das wiederum als „einfacher“ Service in das Repository eingestellt wird. Eine weitere Annahme des Mashup-Modells bezieht sich auf die Ausführung der das Mashup bildenden Dienste. Diese können sowohl sequentiell als auch parallel ausgeführt werden. Außerdem lässt sich ein Service-Mashup in verschiedene atomare Einheiten aufteilen. Diese bestehen aus Services, die zusammengenommen eine Leistung erbringen, die der Mashup-Ersteller gegenüber dem Nutzer abrechnen kann.
Die Anforderungen an einen Lösungsansatz für unser Problem lassen sich wie folgt definieren. Als wichtigstes Kriterium sollte natürlich die Problemstellung durch die Lösung behandelt werden, das heißt in unserem Fall dürfen nur erbrachte Leistungen dem Nutzer berechnet werden. Dies sollte einhergehen mit so wenigen Nachteilen wie möglich für die Ausführung der Mashup-Anwendung. Vor allem hinsichtlich der Performanz ist eine geringe Latenz des Ansatzes erstrebenswert, da sonst der grundlegende Vorteil eines REST-basierten Mashups, nämlich die Geschwindigkeit, verspielt wird. Des Weiteren sollte der Lösungsweg robust hinsichtlich der nebenläufigen Ausführung eines Mashups sein, da eine parallele Ausführung von Diensten in einem Mashup möglich ist. Außerdem kann ein Nutzer auch mehrere Mashups auf einmal aufrufen.

1.2 Lösungsansatz

Aufbauend auf den getroffenen Annahmen und den genannten Anforderungen soll nun im Folgenden die Lösungsidee in ihren Grundsätzen skizziert werden.
Eine der grundlegenden Ideen des nachfolgenden Lösungsansatzes liegt in der Vermeidung oder Behandlung von Fehlern **vor** der eigentlichen Ausführung eines Service-Mashups. Hierfür werden vor der abrechnungsrelevanten Ausführung verschiedene Parameter überprüft. Dazu gehören unter anderem das Guthaben des Nutzers und die Erreichbarkeit des Mashups beziehungsweise der benötigten Services.
Dies soll dazu führen, dass Fehlerfälle, die diese Bereiche betreffen, von vorhinein ausgeschlossen oder zumindest minimiert werden.

Die zweite grundlegende Idee beschreibt das Verhalten bei Auftreten eines Fehlers während der Ausführung des Mashups. In diesem Fall wird versucht die atomaren Teile des Service-Mashups abzubrechen beziehungsweise zurückzufahren, so dass ein konsistenter Zustand hinsichtlich der Leistungserbringung und der dazugehörigen Abrechnung hergestellt werden kann.
Die Umsetzung dieser Ideen soll durch ein Interaktionsprotokoll geschehen, welches den Nachrichtenaustausch zwischen Mashup-Nutzer, Mashup und den beteiligten Services so koordiniert, dass obige Ziele erreicht werden. Das Protokoll soll zudem das den RESTful Services zugrunde liegende HTTP-Protokoll erweitern, um die Komplexität durch Hinzunahme einer weiteren Protokollschicht und die damit einhergehenden Geschwindigkeitseinbußen zu vermeiden. Dazu sollen die HTTP-Erweiterungsmöglichkeiten hinsichtlich der Status-Codes, der Methodenspezifikation und der Definition eigener Header genutzt werden.

1.3 Vorgehensweise

Als nächstes folgt nun ein Überblick über die Vorgehensweise bei dieser Arbeit und die nachfolgenden Kapitel.
Zunächst werden in Kapitel 2 die Grundlagen zum Thema Mashup vorgestellt (Kapitel 2.1). Danach erfolgt die Darstellung der Abrechnung bei kostenpflichtigen Diensten und der dabei auftretenden Fehlerszenarien (Kapitel 2.2). Nachdem die Fehlerfälle auf Anwendungseben vorgestellt worden sind, folgt eine Analyse auf technischer Ebene. Hierzu werden die Mashup-Anwendungen zunächst als verteiltes System modelliert (Kapitel 2.3) und die bei einem solchen System auftretenden Fehlerfälle auf den Mashup-Kontext bezogen. Danach wird überprüft, inwieweit die den REST-basierten Mashups zugrunde liegenden Protokolle diese Fehlerszenarien behandeln (Kapitel 2.4). In Kapitel 2.5 werden Transaktionen als ein möglicher Lösungsansatz vorgestellt. Abschließend erfolgt eine Zusammenfassung und Diskussion der Fehlerfälle auf technischer und anwendungsorientierter Ebene (Kapitel 2.6).
In Kapitel 3 wird die Spezifikation des Interaktionsprotokolls vorgestellt. Hierbei bilden die in Kapitel 3.1 vorgestellten Anforderungen den Ausgangspunkt für die Darstellung des Protokollablaufes (Kapitel 3.2). Die dem zugrunde liegenden Konzepte werden in Kapitel 3.3 vorgestellt. Als letztes wird die Behandlung der in Kapitel 2 erarbeiteten Fehlerfälle durch das Interaktionsprotokoll gezeigt (Kapitel 3.4).
Nachdem die Spezifikation des Protokolls vorgestellt worden ist, erfolgt in Kapitel 4 die Umsetzung eines Prototyps. Hierfür werden zunächst die an die Implementierung gestellten Anforderungen dargestellt (Kapitel 4.1). Als nächstes folgt die Vorstellung der dem Prototyp zugrunde liegenden Architektur auf Komponenten- und Modulebene (Kapitel 4.2). Die Module werden dann im darauffolgenden Kapitel 4.3 detaillierter hinsichtlich ihrer Klassen dargestellt. Außerdem beinhaltet dieses Kapitel eine Beschreibung der ver-

wendeten Technologien. Abschließend wird in der Evaluation der Protokollablauf für ein Beispielszenario gezeigt (Kapitel 4.4).
Im letzten Kapitel erfolgt zunächst ein Überblick über das in dieser Arbeit behandelte Problem und den hierfür vorgestellten Lösungsansatz (Kapitel 5.1). Danach wird dieser Ansatz hinsichtlich der Spezifikation und Implementierung diskutiert (Kapitel 5.2). Als letztes wird ein Ausblick gegeben über Themenbereiche und Fragestellungen, die in diesem Zusammenhang noch erörtert werden könnten (Kapitel 5.3).

2 Recherche von Fehlerfällen bei Mashups

Nachfolgend wird nun kurz der Aufbau dieses Kapitels erläutert. Als erstes werden Mashup-Anwendungen im Allgemeinen vorgestellt. Das heißt es wird diskutiert, was der Begriff „Mashup“ bedeutet, die Vor- und Nachteile solcher Service-Mashups, sowie ihre Architektur und verwendeten Technologien.
Danach folgen die Einführung der Bezahlung von Elektronischen Services mit Hilfe eines Prepaid-Systems und die Untersuchung der dabei auftretenden Fehlerfälle.
Nachdem die anwendungsorientierten Fehlerfälle vorgestellt worden sind, wird in den nächsten Abschnitten das Szenario von technischer Seite betrachtet. Hierfür werden die Service-Mashups als ein verteiltes System modelliert und hinsichtlich auftretender Fehler in einem solchen System untersucht.
Danach liegt der Fokus auf dem REST-Architekturstil. In diesem Abschnitt werden nach der Vorstellung der grundlegenden Prinzipien von REST das Internet als konkrete Implementierung und seine Protokolle vorgestellt. Des Weiteren werden mögliche Lösungsansätze für die erarbeiteten Probleme diskutiert, vor allem hinsichtlich der Zuverlässigkeit des HTTP-Protokolls.
Danach werden Transaktionen als ein möglicher Lösungsbestandteil, zuerst im Allgemeinen und danach in Bezug auf das Anwendungsprotokoll vorgestellt und diskutiert.
Als letztes erfolgen eine Zusammenfassung hinsichtlich der herausgearbeiteten Fehlerfälle aus anwendungsorientierter und technischer Sicht und die Diskussion ihrer Lösungsmöglichkeiten.

2.1 Mashup-Anwendungen

Der Begriff „Mashup“ kommt von englisch „to mash“ für vermischen und beschreibt einen Weg, um neue Web-Anwendungen durch Kombination bereits vorhandener Web-Ressourcen und unter Zuhilfenahme von Web-APIs zu erstellen. [vgl. BeDS08; S. 13] APIs sind hierbei offene Programmierschnittstellen, die heutzutage von vielen Webseiten und Webanwendungen angeboten werden und bieten einen standardisierten Zugriff auf Daten. Dadurch ist eine leichte Entwicklung von Mashups möglich. Da diese meistens im Browser des Nutzers laufen, fallen zudem Probleme weg, die zum Beispiel die Installation oder Treiber betreffen. Dies ermöglicht damit auch das Entwickeln von Mashup-Anwendungen mit geringen Programmierkenntnissen. Unter Umständen ist es sogar möglich, das Mashup in einem grafischen Editor „zusammen zu klicken“. [vgl. CCHZ08; S. 19-20]
Ein Nachteil beim Programmieren eines Mashups ist die Abhängigkeit vom Anbieter der Schnittstelle. Diese wirkt sich dahingehend negativ aus, dass die Ausgestaltung der Schnittstelle in der Regel vom Benutzer nicht beein-

flusst werden kann. Außerdem wirken sich Änderungen in der Definition der Schnittstelle ebenfalls schlecht auf ein Mashup aus. Da der Anbieter einer solchen API an der Benutzung seiner Schnittstelle interessiert ist, wird er aus Eigeninteresse versuchen, obige Probleme selber zu vermeiden.
Diese Tatsache führt zusammen mit den Vorteilen bei der Erstellung und Nutzung, vor allem dem generierten Mehrwert, zu einer wachsenden Beliebtheit von Mashup-Anwendungen.

2.1.1 Architektur

Im Folgenden wird die grundlegende Architektur von Mashup-Anwendungen vorgestellt.

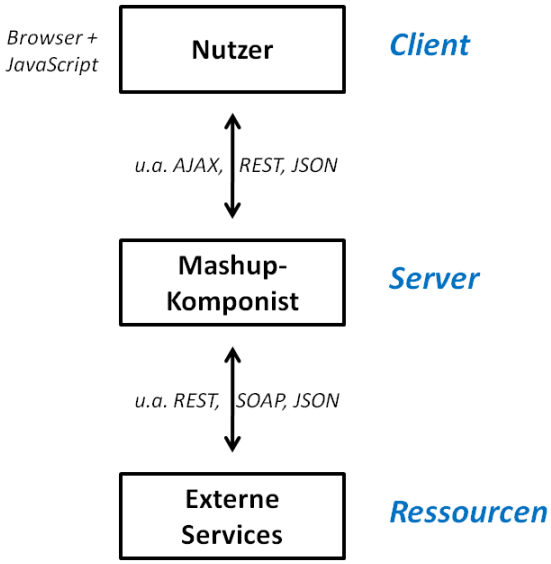

Abbildung 1: Mashup-Architektur

Wie Abbildung 1 zeigt, sind bei einer Mashup-Anwendung die drei Rollen, Client, Server und Ressource, beteiligt.
Der Client ist dabei der Nutzer des Mashups, das in seinem Browser dargestellt wird. Führt der Browser zudem per JavaScript Teile des Mashups selber aus, so spricht man von einem Client-seitigen Mashup. Dem gegenüber stehen die Server-seitigen Mashup-Anwendungen, die vom Server ausgeführt werden. Dabei greift dieser auf Ressourcen aus dem Internet zurück, die die Bestandteile seiner Mashup-Anwendung darstellen.
Die Pfeile in Abbildung 1 stellen die Netzwerkkommunikation dar. Diese erfolgt in allen Fällen über das HTTP-Protokoll. Die benutzten Kommunikationsmuster sind dabei REST und AJAX. JSON und SOAP beschreiben die Nachrichtenformate.

2.1.2 Technologien

Ein wichtiges Charakteristikum von Mashup-Anwendungen ist der Zugriff auf Daten oder externe Anwendungen über das Web. Dabei muss spezifiziert werden, wie die offenen Schnittstellen angesprochen, sowie die Formen, in denen die angeforderten Daten an das Mashup zurückgeben werden sollen. Beginnen möchte ich im Folgenden mit den Anforderungsformaten. Dabei werden die beliebtesten Technologien für den Zugriff auf Ressourcen kurz näher vorgestellt und ihre Funktionsweise innerhalb einer Mashup-Anwendung erläutert.

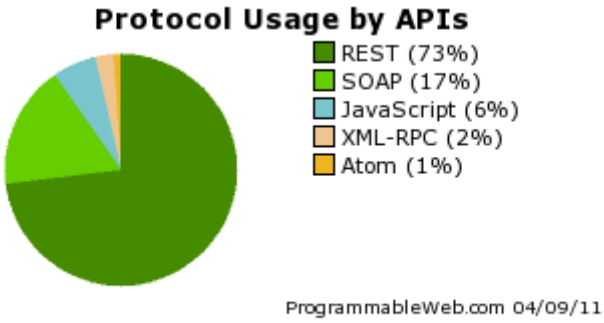

Abbildung 2: beliebteste API-Technologien

Anforderungsformate:

JavaScript

JavaScript ist eine Skriptsprache, die im Wesentlichen im World Wide Web genutzt wird. Sie lässt sich beispielsweise für die Kontrolle von Bestandteilen einer Webseite, Java-Applets und anderen Objekten verwenden. JavaScript ist dabei eine objektbasierte, aber keine objektorientierte Sprache. [vgl. Stey10; S. 36]
Im Kontext eines Mashups besteht durch die Nutzung einer JavaScript-API die Möglichkeit auf Daten oder andere Ressourcen zuzugreifen. Hierbei wird eine JavaScript-Datei in das Mashup eingebunden und damit der Zugriff auf Objekte, Funktionen und Variablen mit denen die externe Anwendung gesteuert werden kann, möglich. Dabei läuft die meiste Arbeit clientseitig ab, da JavaScript vor allem im Webbrowser ausgeführt wird. [vgl. CCHZ08; S. 32]

SOAP

SOAP ist ein auf XML-basierendes Kommunikationsprotokoll zum Austausch von Nachrichten zwischen Computern, unabhängig von ihren Betriebssystemen oder Programmierumgebungen. [vgl. Papa07; S. 120] Dabei wird in den meisten Fällen HTTP als zugrundeliegendes Transportprotokoll genutzt.

Bei einer Mashup-Anwendung kann durch das Senden einer SOAP-Request-Nachricht an eine SOAP-Schnittstelle auf Ressourcen in der Cloud zugegriffen werden. Dabei erfolgt die Verarbeitung der Anfrage beim aufgerufenen Service und nicht wie bei JavaScript bei der aufrufenden Anwendung.

REST

Nach Abbildung 2 werdend die meisten Schnittstellen in Form von REST-APIs zur Verfügung gestellt beziehungsweise genutzt. Der Begriff REST steht für „REpresentational State Transfer“ und stammt aus der Dissertation von Roy Fielding aus dem Jahre 2000. REST beschreibt einen Architekturstil für die Entwicklung von lose gekoppelten Systemen, dessen konkrete Umsetzung zum Beispiel das World Wide Web mit dem HTTP-Protokoll darstellt. [vgl. Tilk09; S. 7,8]
Mit Hilfe eines HTTP-Request kann ein Mashup im Web über eine REST-konforme Schnittstelle auf Daten oder eine externe Anwendung zugreifen und erhält die angeforderten Daten in einer HTTP-Response-Nachricht zurück. Wie auch bei SOAP findet die Bearbeitung auf der Seite des Schnittstellen-Anbieters statt und es werden lediglich die generierten Daten zurückgegeben.
Im Rahmen des Cocktail-Projektes wurde REST als Anforderungsformat ausgewählt und bildet damit die technische Grundlage bei der Entwicklung des Interaktionsprotokolls. Daher erfolgt im Verlauf dieses Kapitels eine tiefergehende Darstellung diese Technologie.

Ausgabeformate:

Nachdem die wichtigsten Anforderungsformate kurz vorgestellt worden sind, werden nun die verschiedenen Formen für die Ausgabe von Daten angesprochen.

SOAP-Response

Handelt es sich bei der angesprochenen Schnittstelle um eine SOAP-API, so erfolgt als Rückgabe eine SOAP-Response, in der die angeforderten Daten enthalten sind. Diese Nachricht ist gemäß der SOAP-Spezifikation aufgebaut und muss von einer dieses Protokoll implementierenden Gegenstelle angenommen und interpretiert werden.

XML

Mit der „Extensible Markup Language“ (XML) wurde vom World Wide Web Consortium eine Sprache zur logischen Dokumentauszeichnung standardisiert. XML ist ein einfaches, universales Format für den Austausch strukturierter Informationen. Außerdem ist XML eine Metasprache, mit der sich

Sprachen für konkrete Anwendungssituationen entwerfen lassen. [vgl. WKBJ03; S. 16-18]

JSON

Die „JavaScript Object Notation“, kurz JSON, ist wie XML ein maschinenlesbares Klartextformat zum Datenaustausch, das zwar weniger flexibel als XML ist, aber viel kompakter. Außerdem beinhaltet JSON direkt an JavaScript orientierte Datenstrukturen beziehungsweise. Datentypen, die verschachtelt und beliebig durch Whitespace-Zeichen strukturiert werden können. Der Vorteil von JSON liegt darin, dass es einfach ist und von vielen Programmiersprachen unmittelbar verarbeitet werden kann. [vgl. Stey10; S. 751-753]

2.2 Metering bei E-Services

In diesem Abschnitt wird zunächst die Bezahlung bei elektronischen Diensten und konkret das Metering vorgestellt. Danach erfolgt eine Analyse hinsichtlich der auftretenden Fehlerfälle.

2.2.1 Allgemeines

Da es bei den hier durchgeführten Untersuchungen um kommerzielle Mashup-Anwendungen geht, wird in diesem Abschnitt der Bezahlvorgang bei elektronischen Services untersucht. Es steht die Frage im Vordergrund, wie die Bezahlung im Internet realisiert werden kann. Eine mögliche Zahlungsform im Internet ist dabei die Nutzung eines elektronischen Bezahlverfahrens. Dieser Begriff bezeichnet „Verfahren, die es ermöglichen, für den Bezug von Gütern und Leistungen eine Gegenleistung über elektronische Netzwerke zu erbringen und deren Ziel allein die Herstellung der Zahlungsfähigkeit von Wirtschaftssubjekten ist.“ [vgl. DaUl04; S. 27] Hierbei stehen verschiedene grundsätzliche Formen zur Auswahl, wie zum Beispiel die Abwicklung der Bezahlung über ein Inkassosystem beziehungsweise mit Hilfe von Verfügungsinstrumenten über Bankkonten. [vgl. DaUl04; S. 37] Wir wollen uns im weiteren Verlauf auf das Prepaid-Zahlungssystem konzentrieren, da dies im Rahmen des Cocktail-Projektes untersucht werden soll. Außerdem ist ein solches System gut geeignet, um geringe Beträge, wie sie bei elektronischen Diensten vor allem anfallen, abzurechnen. Dieses Prepaid-Bezahlsystem wird von der Plattform des Cocktail-Projektes implementiert, das heißt, sie übernimmt die Verwaltung der Prepaid-Konten und das Mitprotokollieren der Service-Aufrufe. Bezogen auf die Problemstellung hinsichtlich der fehlerfreien Abrechnung von Service-Aufrufen steht daher das korrekte Metering der Aufrufe im Vordergrund und nicht der vollständige Bezahlvorgang.

2.2.2 Fehlerfälle bei Prepaid-Zahlungssystemen

Im Folgenden geht es darum, Fehlerfälle bei der Abrechnung von Mashup-Anwendungen herauszuarbeiten. Grundsätzlich lassen sich zwei Fehlerarten bei einem Prepaid-System unterscheiden. Einerseits den Fall, bei dem der Kunde eine Leistung anfordert, ohne ein ausreichendes Guthaben zu besitzen und als zweites die Situation, in der der Nutzer die Zahlung getätigt hat, aber nicht die angeforderte Gegenleistung geliefert bekommt.
Im Folgenden werden diese zwei grundlegenden Situationen auf den Mashup-Kontext bezogen und die spezifischen Fehler aufgezeigt. Dabei erfolgt zunächst die Vorstellung von Fehlerfällen auf Anwendungsebene eines Mashups beziehungsweise der zuarbeitenden Services. Die technischen Fehlerarten werden dann in Kapitel 2.3 und 2.4 erarbeitet.
Als erstes wird das Problem des fehlenden Guthabens erläutert. Hier kann es passieren, dass der Nutzer bei Aufruf des Mashups nicht genügend Geld auf seinem Prepaid-Konto besitzt und damit den Aufruf nicht bezahlen kann. Außerdem besteht, wie im ersten Kapitel erwähnt, die Möglichkeit, dass ein Dienst mit einem dynamischen Preis ausgezeichnet ist, der sich erst zur Laufzeit bestimmen lässt. Dadurch entsteht ein Widerspruch zwischen der Guthabenprüfung ex-ante und des ex-post vorliegenden Preises.
Das zweite grundlegende Problem bezüglich der nicht erfolgten Lieferung bei vorheriger Bezahlung der Leistung kann bei einer Mashup-Anwendung oft vorkommen, da diese aus verschiedenen externen Diensten besteht, die jeder für sich ausfallen können. Außerdem kann ein solcher Service ebenfalls ein Mashup darstellen, wodurch die hier diskutieren Fehlerfälle auch für eine einzelne Service-Komponente gelten können. Diese fehlerhafte Lieferung kann mehrere Ursachen haben. Zum einen ist es vorstellbar, dass der Nutzer die Mashup-Anwendung mit falschen Eingabeparametern aufruft, wodurch es zu einer fehlerhaften Ausführung kommen kann. Auch ist es möglich, dass der Mashup-Dienst beziehungsweise einer seiner zuliefernden Services nicht verfügbar ist. Ein weiteres Problem dieser Fehlerklasse besteht darin, dass es möglich ist, dass für den Mashup-Provider aufgrund von Service-Aufrufen Kosten anfallen, er aber aufgrund eines Fehlers die vom Nutzer geforderte Leistung nicht erfüllen kann und damit keine Kompensation erhält. Hierbei muss die Ursache des Fehlers nicht bei ihm selber liegen, sondern kann zum Beispiel durch einen externen Service verursacht werden.
Die letzte Fehlerklasse unterstellt keine Fehler bei der Beziehung zwischen Bezahlung und Lieferung, sondern beim Bezahlvorgang, also dem korrekten Metering der Mashup-Ausführung. Hier kann es zu Problemen kommen, da innerhalb eines Mashups Dienste parallel aufgerufen werden können und dies zu einem fehlerhaften Logging des Abrechnungsvorganges führen kann. Außerdem ist es möglich, dass der Nutzer mehrere Mashup-Dienste gleichzeitig aufruft und diese gleichzeitig auf dessen Guthabenkonto zugreifen möchten. Auch dies kann zu einer inkonsistenten Abrechnung führen.

2.3 Fehlerfälle in verteilten Systemen

Tanenbaum definiert ein verteiltes System als „eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.“ Wichtig ist herbei, dass „ein verteiltes System aus Komponenten (also Computern) besteht, die autonom sind.“ [vgl. TaSt08; S. 19] Dies ermöglicht eine nebenläufige Ausführung von Programmen und macht das System fehlertoleranter, da bei dem Ausfall einer Komponente die anderen nicht notwendigerweise betroffen sind. Jedoch weisen diese Charakteristika ihre eigenen Probleme und Fehlerfälle auf. So kann es bei der gleichzeitigen Ausführung von Programmen zu Konflikten beim Zugriff auf gemeinsame Ressourcen kommen. Außerdem kann ein Teilausfall auch die Funktionsfähigkeit des gesamten Systems beeinträchtigen. Daher folgt nun im nächsten Abschnitt eine Analyse von Fehlerfällen in verteilten Systemen.

2.3.1 Fehlerfälle

Zunächst wird definiert, was mit dem Ausdruck „Fehler“ beziehungsweise „Fehlerfall“ in unserem Zusammenhang gemeint ist. Dazu muss differenziert werden zwischen den Begriffen Fehlerzustand (Error), Störung (fault) und Ausfall (failure). Ein System fällt aus, wenn es seine Zusagen nicht einhalten kann. Eine solche Zusage könnte zum Beispiel ein Dienst sein, der Pakete aus dem Netzwerk entgegennimmt. Dieser Dienst kann ausfallen, wenn das Übertragungsmedium einer Störung unterliegt. Diese Störung ist damit die Ursache des Ausfalls. Ein beschädigtes Paket in diesem Fall wäre ein Fehler (Error) und damit ein Teil des Systemzustandes, der zu einem Ausfall führen kann. [vgl. TaSt08; S. 356]
Da es in einem verteilten System viele verschiedene Fehlerfälle gibt, wird im Folgenden eine Klassifizierung von Ausfallarten vorgestellt (Abbildung 3). Eine Erläuterung erfolgt im nächsten Abschnitt mit Bezug auf das Mashup-Szenario.

Ausfallart	Beschreibung
Absturzausfall (Crash Failure)	Ein Server steht, hat aber bis dahin richtig gearbeitet. Der angebotene Dienst bleibt beständig aus (ständiger Dienstausfall).
Dienstausfall (Omission Failure) *Empfangsauslassung* *Sendeauslassung*	Ein Server antwortet nicht auf eingehende Anforderungen. Ein Server erhält keine eingehenden Anforderungen. Ein Server sendet keine Nachrichten.
Zeitbedingter Ausfall (Timing Failure)	Die Antwortzeit eines Servers liegt außerhalb des festgelegten Zeitintervalls.
Ausfall korrekter Antwort (Response Failure) Ausfall durch *Wertfehler (Value Failure)* Ausfall durch *Zustandsübergangsfehler (State Transition Failure)*	Die Antwort eines Servers ist falsch. Dieser Ausfall wird oft auch kurz Antwortfehler genannt. Der Wert der Antwort ist falsch. Der Server weicht vom richtigen Programmablauf ab.
Byzantinischer oder zufälliger Ausfall (Arbitrary oder Byzantine Failure)	Ein Server erstellt zufällige Antworten zu zufälligen Zeiten.

Abbildung 3: unterschiedliche Fehlerarten [TaSt08; S. 357]

2.3.2 Mashup-Anwendungen als verteiltes System

Nach obiger Definition in 2.3.1 besteht ein verteiltes System aus unabhängigen Komponenten, die für den Benutzer als **ein** System aussehen. Bezogen auf den Mashup-Kontext, stellt dieses eine System für den Nutzer das ausführbare Mashup dar. Des Weiteren entsprechen die verschiedenen Services den Komponenten und da diese von unterschiedlichen externen Anbietern stammen, sind sie auch meistens unabhängig. Insofern lässt sich eine Mashup-Anwendung als verteiltes System ansehen und damit lässt sich die in Abbildung 3 vorgestellte Klassifizierung anwenden.
Nachfolgend werden daher die aus Abbildung 3 bekannten Ausfallarten auf den Mashup-Kontext bezogen und mit den bereits herausgearbeiteten Fehlerklassen verglichen.
Ein Absturzausfall kann bei einem Mashup-Aufruf zum Beispiel durch Aufhängen des Betriebssystems eines externen Dienstes zustande kommen, so dass dieser keine Antwortnachricht an den Mashup-Komponisten schicken kann. Aufgrund dieser Eigenschaft lässt sich dieser Fehlerfall nicht auf die bisherigen Fehlerklassen abbilden, da diese von einer zuverlässigen Kommunikation ausgehen.
Ebenfalls keine Antwortnachricht erfolgt bei einem Dienstausfall. Bei einer Mashup-Anwendung könnte dies dadurch geschehen, dass der Server die Nachricht nicht bekommen hat, zum Beispiel durch einen Netzwerkfehler. Der Unterschied zum vorherigen Fehlerfall besteht in der zeitlichen Dauer des Ausfalls. In diesem Fall wird angenommen, dass der Ausfall nur temporär und nicht von permanenter Natur ist.

Bei einem zeitbedingten Ausfall antwortet der Server zu spät. Dies könnte zum Beispiel durch eine Überlastung des Netzwerkes passieren, das die Services untereinander verbindet. Bei diesem Fall wird nach einem Timeout davon ausgegangen, dass keine Antwort erstellt wurde.
Somit haben diese drei ersten Ausfallarten von verteilten Systemen die gemeinsame Eigenschaft, dass auf Kommunikationsebene keine Antwortnachricht vom aufrufenden Client empfangen wird und damit ein neuer Fehlerfall in unserem Zusammenhang vorliegt. Wie dieser von den zugrundeliegenden Protokollen behandelt wird, zeigt das nächste Kapitel.
Die nächste Fehlerklasse beschreibt einen Antwortfehler. Bei einer Mashup-Anwendung könnte dies zum Beispiel dann der Fall sein, wenn ein beteiligter Service ausfällt und damit das Ergebnis des Mashups falsch wird. Dieser Fall wurde schon bei der Untersuchung der anwendungsorientierten Fehler herausgearbeitet und lässt sich damit auf die Situation der „fehlerhaften Lieferung abbilden“.
Als letztes ist der zufällige Ausfall zu nennen. Dieser beinhaltet alle Fehlermöglichkeiten, die von den anderen Klassen nicht abgedeckt werden. Da hier eine Vielzahl an Varianten denkbar sind, soll diese Klasse im weiteren Verlauf nicht weiter berücksichtigt werden.

2.4 REST

Wie in der kurzen Einführung aus Kapitel 2.1.2 angesprochen, stammt der Begriff REST aus der Dissertation von Roy Fielding und steht für „REpresentational State Transfer“. REST ist dabei ein Architekturstil, dessen konkrete Ausprägung das World Wide Web darstellt und der auf den folgenden fünf Kernprinzipien beruht.

Ressourcen mit eindeutiger Identifikation

Eine Ressource stellt in REST eine Abstraktion von einer Information dar. Jegliche benennbare Information kann eine Ressource sein, wie zum Beispiel Dokumente, Bilder,… [vgl. Field00; S. 88] Diese Ressource soll nach REST eindeutig zu adressieren sein.
Dieses Konzept wird im World Wide Web durch die Uniform Ressource Identifier (URI) umgesetzt, die einen globalen Namensraum bilden, in dem jede Ressource eindeutig adressiert werden kann.

Hypermedia

Hinter diesem Begriff verbirgt sich das Konzept von Verknüpfungen, die verschiedene Ressourcen miteinander verbinden. Außerdem steuern sie den Applikationsfluss über die möglichen neuen Zustände.
Im World Wide Web sind diese Verknüpfungen als Links aus HTML bekannt.

Standardmethoden

Um zu spezifizieren was mit der angesprochenen Ressource gemacht werden soll, muss die entsprechende Methode aufgerufen werden. Bei REST implementiert jede Ressource die gleiche Schnittstelle.
Im Web werden vor allem die Methoden GET und POST des HTTP-Protokolls genutzt, GET zum Anfordern einer Repräsentation und POST meistens zum Erstellen einer neuen Ressource.

Ressourcen und Repräsentationen

Da Ressourcen ein abstraktes Konzept sind, können sie nicht direkt verarbeitet werden. Hierfür gibt es die Repräsentationen einer Ressource, die zurückgegeben werden, wenn ein Client eine Ressource anfordert. Dabei kann der Nutzer auswählen, welche der verfügbaren Darstellungen er konsumieren möchte.
Typische Ressourcen-Repräsentationen im Web basieren meistens auf HTML oder XML.

Statuslose Kommunikation

Dieses letzte Prinzip bedeutet nicht statuslose Applikationen. Es besagt stattdessen, dass der Status entweder vom Client gehalten oder in einen Ressourcenstatus umgewandelt werden soll.
Im Web kann eine Anwendung zum Beispiel den Status eines Einkaufvorgangs entweder als Teil der Repräsentation mitschicken oder die Ressource „Warenkorb“ für den entsprechenden Nutzer anlegen. [vgl. Tilk09; S. 9-15]

Vorteile:

Die statuslose Kommunikation bringt vor allem den Vorteil mit sich, dass Anfragen nicht vom gleichen Server bearbeitet werden müssen und damit das System gut skalierbar ist. Positiv ist auch die gute Wiederverwendbarkeit durch die uniforme Schnittstelle. Diese ermöglicht ebenfalls eine lose Kopplung zwischen verschiedenen Systemen und damit eine höhere Flexibilität.
Die konkrete Ausprägung von REST in Form des World Wide Web ermöglicht zusätzlich noch eine hohe Interoperabilität, da der Geltungsbereich der Web-Standards – HTTP, URIs, XML/ HTML – ziemlich weitreichend ist. Außerdem ist das Web ziemlich performant aufgrund der Nutzung von URIs, die das Caching von Dateien erlaubt. Zudem ist durch die Benutzung von nur vier generischen Methoden die Benutzung relativ einfach. [vgl. Tilk09; S. 2-4]

2.4.1 Protokolle des World Wide Web

In diesem Abschnitt werden die für die erarbeiteten Fehlerfälle relevanten Protokolle TCP und HTTP kurz erläutert und hinsichtlich ihrer Behandlung der Fehlerklassen untersucht. Dabei liegt der Fokus vor allem auf dem Anwendungsprotokoll HTTP.

TCP-Protokoll

„Das Transmission Control Protocol (TCP) wurde speziell zur Bereitstellung eines zuverlässigen Bytestroms von einem Endpunkt zum anderen in einem unzuverlässigen Internetwork (Netzverbund) entwickelt.“ [Tane03; S. 580] Wichtig ist hierbei die Eigenschaft der zuverlässigen Übertragung, das heißt, Fehler beim Transport von Endsystem-zu-Endsystem werden behandelt.
Bezogen auf das Mashup-Szenario bedeutet dies, dass vor allem Netzwerkfehler oder allgemeiner Fehler beim Transport der Nachricht in unserem Fall nicht mehr beachtet werden müssen, da TCP eine zuverlässige Übertragung garantiert.

HTTP-Protokoll

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll auf Anwendungsebene für verteilte Hypermedia-Informationssysteme. [vgl. FGMF99; S. 7] Es wird im World Wide Web eingesetzt und ist zustandslos, das heißt es wird kein Zustand gespeichert zwischen den Nachrichtenpaaren. Des Weiteren ist HTTP ein einfaches Anfrage/ Antwort-Protokoll, das bedeutet, auf einen HTTP-Request folgt eine HTTP-Response-Nachricht. Eine HTTP-Nachricht besteht dabei aus zwei Teilen, zum einen dem HTTP-Header, der Metainformationen enthält und zum anderen dem HTTP-Body, der die eigentlichen (Antwort-)Daten beinhaltet. Treten Fehler bei der Bearbeitung einer Anfrage auf, wird dies durch einen entsprechenden HTTP-Status-Code dem Client angezeigt. Dieser Status-Code gibt zudem an, wer oder was den Fehler ausgelöst hat. Zudem können eigene Status-Codes definiert werden. Außerdem spezifiziert HTTP eine Reihe von Verben, die die Schnittstelle bilden. Hierzu zählen unter anderem GET, POST, PUT und DELETE. Dabei können diese Operationen hinsichtlich der Eigenschaft charakterisiert werden, ob sie sicher oder idempotent sind. Eine Methode ist sicher, wenn sie keine Änderungen auslöst. Hierzu zählen GET und HEAD. Eine Methode ist idempotent, wenn das Resultat bei mehrfacher Ausführung gleich dem der einmaligen ist. Zu diesen Operationen zählen, außer GET und HEAD, auch PUT und DELETE. Weitere Eigenschaften von HTTP wie der Aufbau von Verbindungen, das Caching von Dateien und das Aushandeln des Rückgabeformates sollen hier nur kurz erwähnt werden, da sie bei der Entwicklung des Interaktionsprotokolls eine untergeordnete Rolle spielen. [vgl. FGMF99]

2.4.2 Zuverlässigkeit in HTTP

Wichtig in unserem Fall ist die Frage, ob HTTP eine zuverlässige Nachrichtenübertragung von Anwendung zu Anwendung garantieren kann, da sonst unklar ist, wie der Client im Fehlerfall einer nicht erfolgten Antwortnachricht reagieren soll. Das HTTP-Protokoll selbst implementiert keine Mechanismen, die die Zuverlässigkeit der Nachrichtenübertragung sicherstellen können.

Daher werden nachfolgend alternative Möglichkeiten vorgestellt, die zur Erreichung dieses Ziels beitragen können.

Als erstes wird erläutert, wie mit Hilfe idempotenter Methoden das Problem gelöst werden kann. Wie oben schon erwähnt, kann man eine idempotente Operation beliebig oft wiederholen, ohne dass sich das Ergebnis nach der ersten Ausführung ändert. Hierzu zählen bei HTTP unter anderem GET, PUT und DELTE, die normalerweise für das Anfordern und Updaten beziehungsweise Löschen einer Ressource angewendet werden. Die Benutzung von idempotenten Methoden bringt den Vorteil mit sich, dass bei einem Ausbleiben der Antwort vom Server der Client die Anfrage erneut abschicken kann. Tritt der Fehler nach der n-ten Wiederholung immer noch auf, so kann von einem Ausfall des Servers ausgegangen werden. [vgl. Tilk09; S. 156-157] Das Problem dieses Ansatzes ist, dass nicht alle Services ihre Funktionalitäten über idempotente Methoden bereitstellen. Häufig wird die nicht idempotente HTTP-Methode POST benutzt und diese kann bei dem hier vorgestellten Ansatz nicht verwendet werden.

Daher soll nun der POST-Once-Exactly (POE) Ansatz von Mark Nottingham vorgestellt werden, der zum Ziel hat, das Problem der mangelnden Idempotenz von POST zu lösen. Anzumerken ist, dass dieser nicht aus einer offiziellen Spezifikation stammt, sonder lediglich aus einem Arbeitspapier. Das grundlegende Konzept dieses Ansatzes ist die POE Ressource, die die Eigenschaft besitzt, dass sie nur einmal mit POST ausgeführt werden kann. Weitere Versuche auf diese Ressource mit POST zuzugreifen werden mit einem „405: Method not allowed“-Status-Code beantwortet. Ein typischer Anwendungsfall sieht folgendermaßen aus: Der Client gibt durch einen speziellen POE-Header-Eintrag zu erkennen, dass er POE versteht. Die Antwortnachricht seitens des Servers beinhaltet daraufhin einen speziellen POE-Link auf die entsprechende POE-Ressource, den der Client dann ausführen kann. [vgl. Nott05; S. 4-7]

Eine andere Möglichkeit stellt der SOA-Rity Ansatz von Goland dar, dessen grundlegende Idee ist, eine Request-Nachricht mit einer eindeutig identifizierbaren Nummer (ID) und einem Zeitstempel zu versehen. Auch hier ist anzumerken, dass dieser Ansatz aus einem Arbeitspapier und keiner offiziellen Spezifikation stammt. Der bearbeitende Server speichert die ID der ankommenden Nachrichten und kann dadurch zu einem späteren Zeitpunkt nachvollziehen, ob er einen Request schon einmal erhalten hat. Ist das der Fall oder der Zeitstempel nicht mehr gültig, so zeigt er dies dem Client durch einen HTTP-Status-Code an. [vgl. Gola05; S. 4-6]

2.5 Transaktionen

Eine Transaktion bezeichnet eine Menge von Operationen auf dem physischen und abstrakten Zustand der Anwendung […] mit den folgenden Eigenschaften: Atomarität, Konsistenz, Isolation und Dauerhaftigkeit (ACID).

Atomarität meint in diesem Kontext, dass entweder alle Änderungen einer Transaktion ausgeführt werden oder keine. Dazu gehören unter anderem Datenbankänderungen oder Nachrichten.

Konsistenz bedeutet für eine Transaktion, dass diese eine korrekte Transformation des Zustandes darstellt. Die Aktionen, die als Gruppe ausgeführt werden, verstoßen dabei nicht gegen die Integritätsbedingungen.

Isolation besagt, dass trotz der gleichzeitigen Ausführung von Transaktionen es aussieht, als würden sie nacheinander durchgeführt werden.

Dauerhaftigkeit meint, dass nach einer erfolgreichen Transaktion die ausgeführten Änderungen auch im Fehlerfall bestehen bleiben. [vgl. GrRe93; S. 6]

2.5.1 Transaktionen bei Mashup-Anwendungen

Im Folgenden werden die obigen ACID-Eigenschaften auf das Mashup-Szenario bezogen und ihre Bedeutung in diesem Kontext erklärt. Hierbei entspricht ein Mashup-Aufruf seitens des Nutzers einer Transaktion, da dieser die Mashup-Anwendung entweder ganz oder gar nicht ausführen will (Atomarität). Das Kriterium der Konsistenz gilt vor allem hinsichtlich des korrekten Logging des Mashup- beziehungsweise Service-Aufrufes und damit dem entsprechenden Datenbankeintrag der Plattform. Des Weiteren ist es erstrebenswert, dass das Metering durch gleichzeitig ausgeführte Transaktionen nicht beeinträchtigt wird (Isolation). Da es sich um Prepaid-Kontoinformationen handelt, sollen diese auch persistent gespeichert werden (Dauerhaftigkeit). Damit sieht man, dass transaktionale Eigenschaften bei einer Mashup-Anwendung erstrebenswert sind.

2.5.2 Transaktionen in REST - HTTP

In diesem Abschnitt wird behandelt, wie Transaktionen im REST- beziehungsweise HTTP-Kontext umgesetzt werden können. Da das HTTP-Protokoll von sich aus keine Transaktionen unterstützt, gibt es eine lebendige Diskussion darüber, wie entsprechende Konzepte aussehen könnten. [vgl. Litt09]
Der hier vorgestellte Ansatz stammt von Pautasso [vgl. PaPa11] und zeigt wie mit Hilfe des Try-Confirm/ Cancel-Musters (TCC) [vgl. Pard09] Atomarität bei der Ausführung von mehreren REST-basierten Web Services garantiert werden kann. Das TCC-Pattern besagt, dass Ressourcen, bevor sie persistent gespeichert werden, zuerst in einen vorläufigen Zustand übergehen (vgl. Abbildung 4). Tritt nun ein Fehler bei der Transaktion auf, so kann durch ei-

nen CANCEL-Request die Transaktion abgebrochen werden. Gleiches geschieht bei einem Timeout.

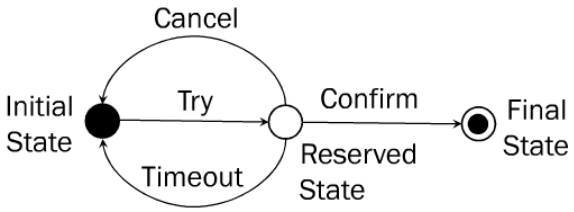

Abbildung 4: TCC-Pattern [PaPa11; S. 6]

Abbildung 5 zeigt die Architektur und den typischen Ablauf für diesen Ansatz. Dabei ruft der Client einen zusammengesetzten REST-basierten Service auf (0). Dieser bewirkt daraufhin bei allen beteiligten Ressourcen durch einen POST-Request (Try-Phase) die Zustandsänderung in den „Reserved State“ (1). Ist diese bei allen Beteiligten erfolgreich verlaufen, so sendet der „Transaction Coordinator“ des Services an alle Ressourcen eine Bestätigung (Confirm-Phase), so dass diese in den finalen Zustand übergehen (4).
Der Vorteil dieses Ansatzes gegenüber anderen Konzepten wie zum Beispiel des RETRO-Modells [vgl. MRMK09] ist die Einfachheit mit der transaktionale Sicherheiten gegeben werden können. Dies ermöglicht eine performante Ausführung, die für Mashup-Anwendungen wichtig ist.

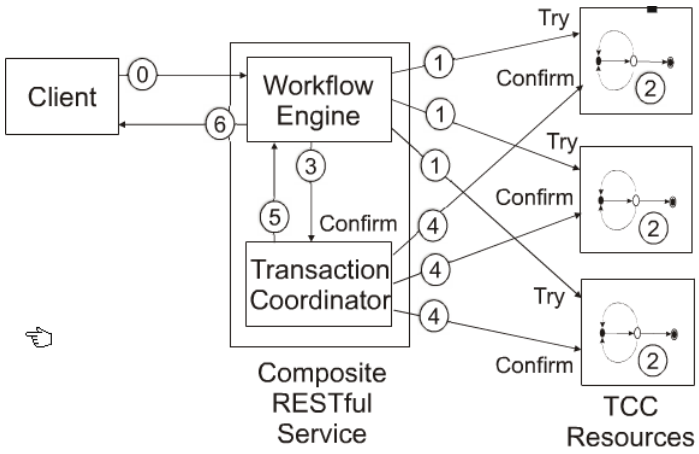

Abbildung 5: Architektur und Ablauf bei einem zusammengesetzten Service [vgl. PaPa11; S. 6]

2.6 Zusammenfassung

Zum Abschluss dieses Grundlagen-Kapitels werden die herausgearbeiteten Fehlerfälle aus anwendungsorientierter und technischer Sicht zusammenfassend dargestellt und die angesprochenen Lösungsmöglichkeiten diskutiert. Auf Basis dessen erfolgt die Definition des Arbeitsauftrages für den im nächsten Kapitel folgenden Lösungsentwurf.

2.6.1 Fehlerszenarien

Fehlerfälle	*Beschreibung*
Nicht ausreichendes Guthaben	Der Nutzer startet ein Mashup, obwohl sein Guthaben dafür nicht ausreicht. Dabei muss unterschieden werden zwischen einem fixen Preis, der im Voraus feststeht und einem variablen Preis, der erst zur Laufzeit komplett bestimmbar ist
Dienst nicht verfügbar	Der Nutzer bezahlt im Voraus für ein Mashup, das nicht verfügbar ist
Falsche Eingabe-Parameter	Der Nutzer startet ein Mashup mit Parametern, die nicht konform gemäß der Spezifikation sind
Fehlerhafte Rückgabe	Der Nutzer startet ein Mashup und bekommt trotz Bezahlung nicht das erwünschte Ergebnis
Parallele Mashup-Aufrufe	Der Nutzer ruft mehrere Mashup-Anwendungen gleichzeitig auf. Dabei kommst es beim Zugriff auf das Konto des Kunden zu Fehlern
Parallele Service-Aufrufe	Eine Mashup-Anwendung ruft parallel mehrere Dienste auf. Dabei kommt es bei der Protokollierung zu Fehlern
Keine Antwortnachricht	Innerhalb der Mashup-Ausführung geht eine Response-Nachricht auf dem Weg zur Anwendung verloren

2.6.2 Diskussion

Das hier grundlegende Problem, nämlich die korrekte Abwicklung der Bezahlung bei Lieferung der Ware, lässt sich als Ausführen einer Transaktion auffassen. Hierbei ist diese nur dann erfolgreich, wenn die Ware korrekt geliefert wird und die Kompensation erfolgt ist. Dies bezieht sich auf die atomare Eigenschaft von Transaktionen. Des Weiteren haben wir es hier mit Nebenläufigkeitsproblemen zu tun, die auch im Kontext von Transaktionen zu finden sind. Diese sollen aber im Folgenden nicht adressiert werden, da hierfür ein anderer Lösungsansatz, wie zum Beispiel Locking-Mechanismen [vgl. GrRe93; S. 449-489], notwendig wäre. Das Interaktionsprotokoll, das im nächsten Kapitel vorgestellt wird, soll sich auf das oben aufgezeigte atomare Problem beziehen.

Die Grundlage für die Spezifikation eines solchen Protokolls ist die Zuverlässigkeit des Nachrichtenaustausches. Hierbei ist vor allem die zuverlässige Übertragung über HTTP interessant, da diese gemäß der Spezifikation nicht unterstützt wird. Die in Abschnitt 2.4.2 vorgestellten Lösungsmöglichkeiten bauen dabei alle auf der gleichen Idee auf, nämlich dem Wiederholen der Anfrage im Fehlerfall und haben alle ihre Stärken und Schwächen. Der erste Ansatz, der das Anwenden von idempotenten Methoden propagiert, bringt den Vorteil mit sich, dass keine zusätzlichen HTTP-Header eingeführt werden oder ein bestimmtes Verhalten eines Dienstes vorausgesetzt wird. Negativ ist aber, dass alle Services ihre Funktionalitäten durch eine idempotente Methode anbieten müssen. Die POST-Once-Exactly Idee umgeht dieses Problem, in dem auch für POST das Wiederholen der Anfrage ohne Nebeneffekte möglich wird. Hierfür muss der anbietende Dienst aber diese Spezifikation implementieren. Gleiches gilt für den SOA-Rity Ansatz, bei dem es durch die Definition zusätzlicher Header ebenfalls möglich wird, POST-Anfragen gefahrlos zu wiederholen. Im weiteren Verlauf soll ein an diese letzte Idee anknüpfender Ansatz verwendet werden. Die detaillierte Vorstellung und Begründung hierfür erfolgt in Kapitel 3.3.1.

Weiterhin ist zu diskutieren, inwiefern die geforderte atomare Ausführung der Transaktion durch eine Implementierung auf Ebene des HTTP-Anwendungsprotokolls erreicht werden kann und ob dies eine Lösung darstellt. Das Hauptproblem bei der Anwendung von Transaktionen auf den Mashup-Kontext ist die gegensätzliche Eigenschaft hinsichtlich der Kopplung der Systeme. Mashups zeichnen sich dadurch aus, dass durch die verschiedenen externen Dienste das System nur lose gekoppelt ist, wohingegen Transaktionen in Systemen angewendet werden, in denen die Komponenten eine starke Abhängigkeit untereinander aufweisen. Somit kann in unserem Fall der in Kapitel 2.5.2 vorgestellte transaktionale Ansatz nicht ohne weiteres auf das Mashup als Ganzes angewendet werden.

3 Lösungsentwurf für das Interaktionsprotokoll

In diesem Kapitel wird der Lösungsentwurf für die in dieser Arbeit behandelten Probleme vorgestellt. Dazu werden als erstes die Anforderungen an die Spezifikation erarbeitet und danach wird der Ablauf des Interaktionsprotokolls bei fehlerfreier Ausführung gezeigt. Im darauffolgenden Abschnitt werden dann die der Protokollspezifikation zugrunde liegenden Ideen erläutert und im letzten Teil erfolgt die Behandlung der im vorherigen Kapitel herausgearbeiteten Fehlerfälle.

3.1 Anforderungen

Die Anforderungen an ein System oder in unserem Fall an ein Protokoll bilden den Ausgangspunkt für die Spezifikation des Entwurfs, da diese Auswirkungen auf grundlegende Eigenschaften des Systems haben. Unterschieden werden kann zwischen funktionalen und nicht-funktionalen Anforderungen. Letztere „sind Beschränkungen der durch das System angebotenen Dienste oder Funktionen“. [vgl. Somm07; S. 152] Auf unseren Kontext bezogen bedeutet dies, dass an das Protokoll weitere über die Behandlung der Fehlerfälle hinausgehende Anforderungen gestellt werden, die vor allem die Einsetzbarkeit der Lösung betreffen. Funktionale Anforderungen „sind Aussagen zu den Diensten, die das System leisten soll, zur Reaktion des Systems auf bestimmte Eingaben und zum Verhalten des Systems in bestimmten Situationen.“ [vgl. Somm07; S. 152] In unserem Fall stellt dies die Fehlerbehandlung durch das Protokoll dar.

3.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen treffen Aussagen zum Verhalten des Systems. Betrachtet man die in Kapitel 2.6.1 und 2.6.2 dargestellten Fehlerfälle, so lassen sich diese dahingehend unterscheiden, dass sie entweder vor oder nach dem Bezahlvorgang eintreten können. Somit können folgende zwei grundlegende funktionale Anforderungen definiert werden.

Behandlung der Fehlerfälle vor der Bezahlung

Diese erste Anforderung fordert das „korrekte“ Verhalten des Systems **vor** dem Bezahlvorgang, bei einem Prepaid-System also vor der Ausführung des Mashups. Konkret bezieht sich diese Anforderung auf den Fehlerfall des nicht ausreichenden Guthabens bei Vorliegen eines fixen Preises.

Behandlung der Fehlerfälle nach der Bezahlung

Diese Forderung bezieht sich auf das Verhalten des Systems **nach** dem Bezahlvorgang. Sie beinhaltet die von der ersten Anforderung nicht erfassten Fehlerszenarien. Hierbei ist es vor allem wichtig, den bereits erfolgten Bezahlvorgang in einen konsistenten Zustand zu überführen.

3.1.2 Nicht-funktionale Anforderungen

Zusätzlich zu den funktionalen Anforderungen werden im Folgenden Beschränkungen für die zu erbringenden Dienste vorgestellt. Diese sind wichtig für den Anwendungsbereich und die Einsetzbarkeit des Interaktionsprotokolls, betreffen aber im Gegensatz zu den funktionalen Anforderungen nicht die Kernfunktionalitäten als solches.

Performance

Damit das Interaktionsprotokoll praktikabel in der Praxis eingesetzt werden kann, darf es die Ausführung einer Mashup-Anwendung nicht zu stark verlangsamen. Daher ist es wichtig bei der Spezifikation einerseits auf einen effizienten Ablauf und andererseits auf wenig zusätzlichen Payload für die Nachrichten zu achten.

Großer Anwendungsbereich

Als letzte Anforderung ist zu nennen, dass die Protokollspezifikation in einem großen Anwendungsbereich einsetzbar ist und nicht nur in einem spezifischen Szenario. Dies ist wichtig, da die Mashup-Provider oder auch die Anbieter externer Dienste ihre Services nicht nur auf einer Plattform bereitstellen wollen, sondern auf mehreren „Vertriebskanälen“.

3.2 Ablauf des Interaktionsprotokolls

Für den Ablauf des Interaktionsprotokolls müssen folgende zwei Szenarien unterschieden werden. Zum einen ist der Preis eines Mashups durch eine im Voraus fixe Komponente festgelegt und zum anderen besteht der Preis aus einem fixen und variablen Teil, wobei letzterer erst zur Laufzeit eindeutig bestimmbar ist.

3.2.1 Fixer Preis einer Mashup-Anwendung

Abbildung 6 zeigt die Abfolge der Nachrichten zwischen Client, Plattform und der Mashup-Anwendung. Hierbei startet der Client, meist aus dem Browser, mit einem HTTP-Request das Mashup (1). Diese Anfrage erfolgt aber nicht direkt an die Mashup-Anwendung, sondern an die Plattform. Diese überprüft vor Ausführung des Mashups das Prepaid-Guthaben des Client (2). Ist dieses ausreichend, so wird die Mashup-Ausführung gestartet (3) und der Preis

vom Konto des Nutzers abgebucht. Der Mashup-Komponist, in unserer Abbildung Service-0, nimmt diesen Request entgegen und überprüft zunächst die Eingabeparameter auf Korrektheit (4). Sind diese konform zur Spezifikation, so schickt er eine Probe-Anfrage an die die erste atomare Einheit bildenden Services (5). In Abbildung 6 stellt diese nur Service-1 dar. Erfolgt eine positive Rückmeldung (6), so wird zuerst eine Statusmeldung an die Plattform zurückgegeben, in der die vorherige Anfrage bestätigt wird (7) und danach der erste Service gestartet (8). Liefert dieser eine korrekte Antwortnachricht zurück (9), so kann der Mashup-Komponist weiter im Ablauf verfahren. In unserer Abbildung heißt das, dass er die zweite und letzte atomare Einheit ausführt. Hierfür sendet er wiederum zunächst eine Probe-Anfrage an Service-2 (10) und Service-3 (12) und bei erfolgreicher Bestätigung seitens der aufgerufenen Dienste (11), (13) werden diese gestartet (14), (16). Bei Beendigung der Mashup-Ausführung übergibt der Mashup-Komponist durch einen Callback-Aufruf an die Plattform die erzielten Ergebnisse (18). Diese bestätigt dann den Eingang der Antwort einerseits hinsichtlich des Abrechnungsvorganges (20) und andererseits gegenüber dem Mashup-Komponisten (19). Danach werden die Ergebnisse an den Client weitergeleitet, meist eingebunden in eine Webseite (21).

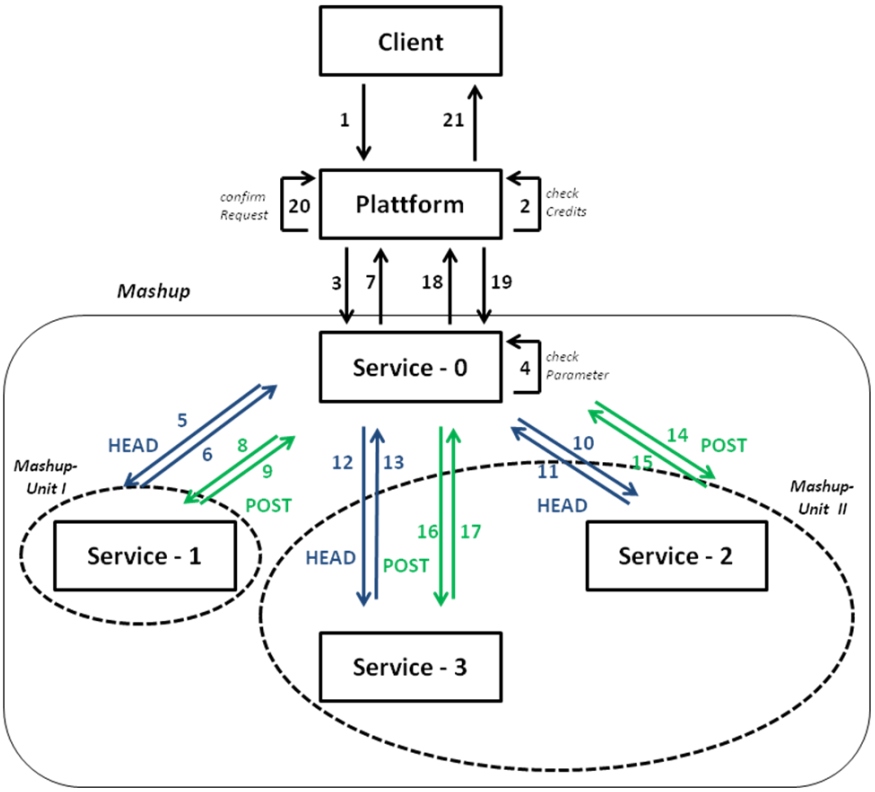

Abbildung 6: Protokollablauf bei fixem Preis

3.2.2 Variabler Preis einer Mashup-Anwendung

In Abbildung 7 wird davon ausgegangen, dass die Mashup-Anwendung durch einen zusammengesetzten Preis beschrieben wird. Dabei könnte es sich zum Beispiel um einen Letter-Shop-Service handeln, bei dem Service-1 Datensätze aus einer CRM-Datenbank zurückgibt, Service-2 diese nach dem Kriterium „Interessent“ filtert und Service-3 an diese Interessenten Werbebriefe verschickt. Service-1 und Service-2 bieten hierbei ihre Dienste für einen fixen Preis an, der Preis von Service-3 ist abhängig von der Anzahl der zu verschickenden Brief und damit den gefilterten Datensätzen. Daher erfolgt hier nachdem die Anzahl der Datensätze feststeht (15), eine zweite Überprüfung des Guthabens des Nutzers. Allgemein ist diese Prüfung an der Stelle in der Ausführung des Mashups vorgesehen, an der der variable Preis bestimmbar ist. Ansonsten unterscheidet sich der Ablauf nicht vom dem bei einem fixen Preis.

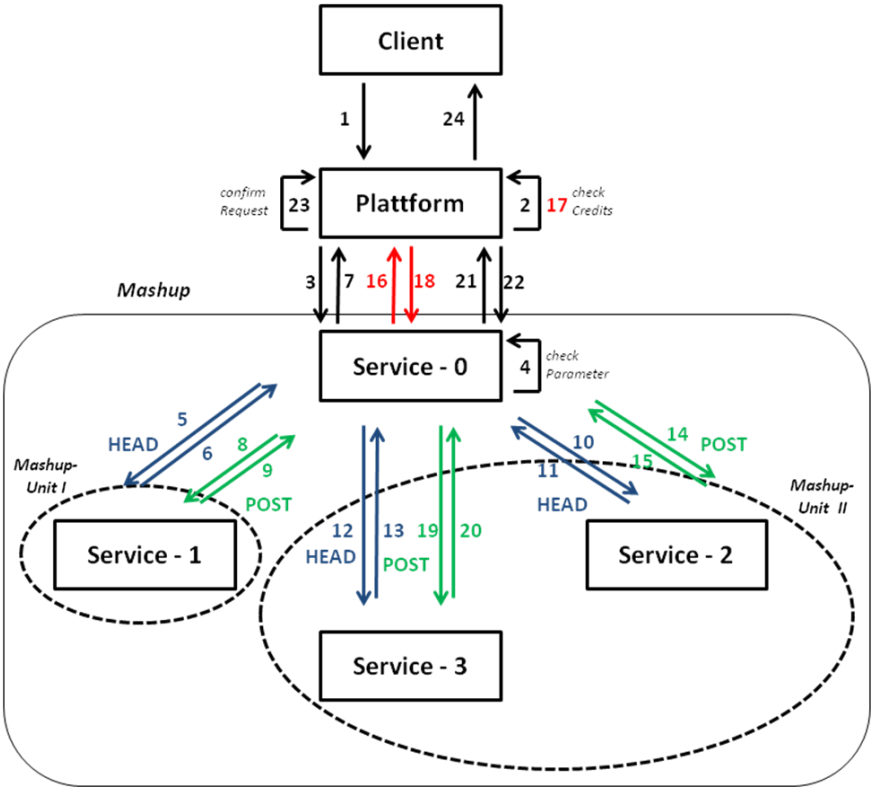

Abbildung 7: Protokollablauf bei variablem Preis

3.3 Grundlegende Konzepte des Interaktionsprotokolls

Nach dem der Ablauf des Interaktionsprotokolls im fehlerfreien Fall vorgestellt worden ist, werden in diesem Abschnitt die Grundlagen der Spezifikation dargelegt.

3.3.1 Zuverlässigkeit auf Kommunikationsebene

Damit die Spezifikation eines Interaktionsprotokolls möglich ist, muss sichergestellt werden, dass der zuverlässige Nachrichtenaustausch garantiert werden kann, oder dass zumindest eine Behandlung im Fehlerfall erfolgt. Dazu wird der in Kapitel 2.6.2 gewählte Ansatz zur Zuverlässigkeit von HTTP detaillierter vorgestellt und begründet.
Die Grundidee besteht darin, im Falle einer nicht empfangen Antwort seitens des Clients den Request zu wiederholen, solange bis eine Response-Nachricht erfolgt ist oder bis eine obere Grenze an Wiederholungen überschritten ist. Es stellt sich die Frage, wie sich der Server verhält, wenn er mehrmals die gleiche Anfrage bekommt. Typischerweise erfolgt bei einem Mashup ein Service-Aufruf mit dem Ziel, bestimmte Daten anzufordern. Dabei soll kein Zustand auf dem Server gespeichert, beziehungsweise keine Ressource angelegt werden. Ein Problem entsteht, wenn ein kostenpflichtiger Dienst die angeforderte Aufgabe mehrmals ausführt und berechnet. Daher soll ein an SOA-Rity [vgl. Gola05] angelehnter Ansatz als Grundlage für das Interaktionsprotokoll verwendet und nachfolgend näher beschrieben werden.
Erhält ein Service auf seine HTTP-Anfrage an einen anderen Dienst keine Antwort innerhalb einer gewissen Zeit, so geht er von einem Fehler aus und wiederholt die Anfrage. Dabei verwendet er für seinen Request einen Message-Identifier, der diesen eindeutig identifiziert. Dies ermöglicht dem Server bei Eintreffen einer Anfrage zu überprüfen, ob er diese schon verarbeitet hat. Ist dies nicht der Fall, so bearbeitet er den Request und sendet in der HTTP-Response Nachricht eine Bestätigung mit. Bekommt er eine Anfrage, die er bereits verarbeitet hat, zeigt er dies dem Client an. (vgl. Abbildung 8)

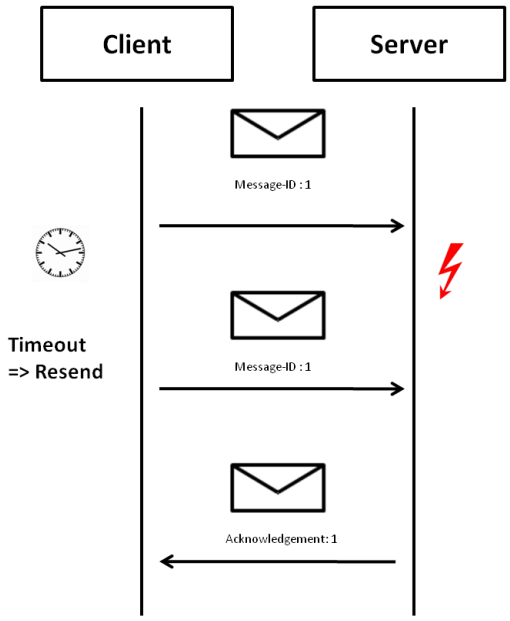

Abbildung 8: Zuverlässigkeit in HTTP

Nachfolgend wird aufgezeigt, wie obiger Mechanismus in das Interaktionsprotokoll integriert ist.
Abbildung 9 zeigt, dass jeder HTTP-Request durch ein Acknowledgement in der entsprechenden HTTP-Response bestätigt wird. Bleibt dieses an Position (6) oder (7) aus, so wird das Problem gemäß der Fehlerklasse „Dienst nicht verfügbar“ behandelt. Fehlt die Bestätigung im weiteren Verlauf, so lassen sich diese Fälle auf die Klasse „Fehlerhafte Rückgabe“ abbilden. Bei Vorliegen des dynamischen Preis-Szenarios erfolgt die Bestätigung in der Response-Nachricht der Plattform (Abbildung 7 (18)).

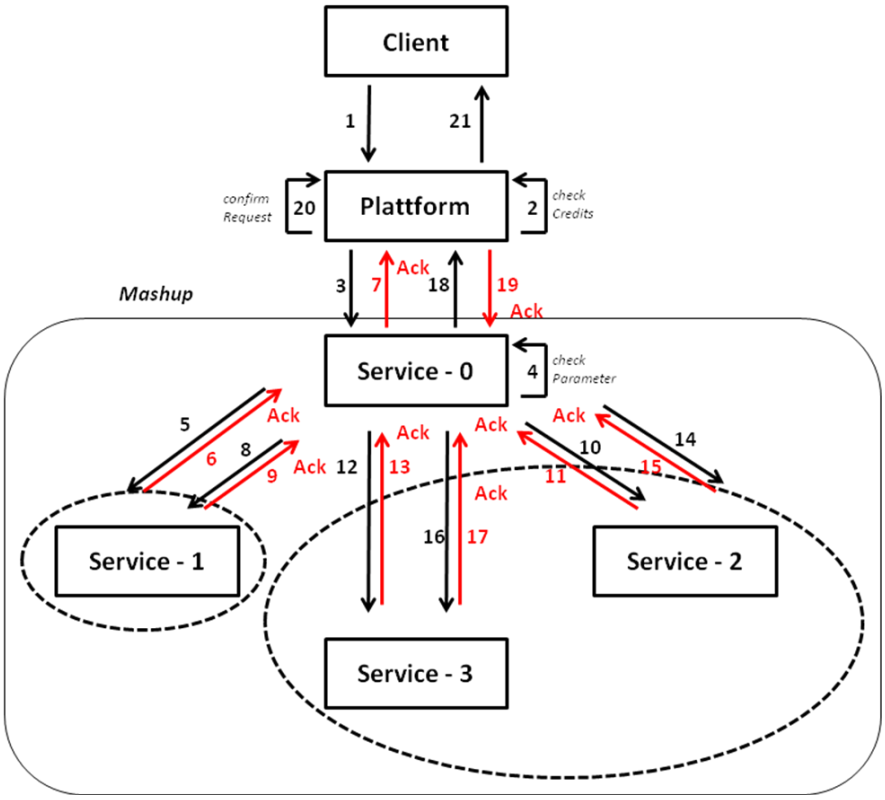

Abbildung 9: Bestätigungen im Protokollablauf

3.3.2 Metering der Mashup-Ausführung

Ein weiterer wichtiger Prozessschritt ist der Abrechnungsvorgang beziehungsweise das Metering bei Aufruf eines Mashups. Hierbei muss einerseits die Kompensation seitens des Clients gegenüber dem Mashup-Ersteller erfolgen und andererseits muss der Mashup-Komponist die externen Services vergüten. Dabei wird im Folgenden davon ausgegangen, dass alle Akteure bei der Plattform registriert sind und ein Konto besitzen.
Konkret läuft der Abrechnungsvorgang wie folgt ab (vgl. Abbildung 10): Startet ein Nutzer eine Mashup-Anwendung, so wird bei Schritt (2) der Preis für die Ausführung, soweit bekannt, von seinem Konto vorläufig abgebucht. Vorläufig heißt in diesem Fall, dass sich die Plattform den Wert des alten Kontostandes merkt, da dieser je nach Erfolg der Ausführung wieder hergestellt

werden muss. Des Weiteren werden auch die Vergütungen für die beteiligten Services vorläufig ausgeführt. Dies setzt voraus, dass die Plattform die Beteiligten Akteure und deren Preise kennt. Da aber alle Services mit ihren Preisen bei der Plattform registriert sind, ist dies vorauszusetzen. Damit gibt es nach (2) für jeden Beteiligten einen entsprechenden positiven beziehungsweise negativen (vorläufigen) Buchungsvermerk in der Datenbank der Plattform. Nach dem Ablauf der Mashup-Anwendung wird das Ergebnis an die Plattform gemeldet, die dieses wiederum an den Client weiterleitet. Ist dieses Ergebnis korrekt, so bestätigt die Plattform den Abbuchungsvorgang (20) und damit verliert dieser seinen vorläufigen Charakter. Wird kein Ergebnis zurückgeliefert, so entfällt die Bestätigung und der vorläufige Kontostand wird wiederhergestellt.

Abbildung 10: Metering im Protokollablauf

3.3.3 Behandlung von Fehlerfällen vor der kostenpflichtigen Ausführung

Eine weitere Idee des Interaktionsprotokolls besteht in der Vermeidung von Fehlern vor der kostenpflichtigen Ausführung einer Mashup-Anwendung. Dazu werden das Guthaben (2), die Eingabeparameter (4) und die Erreichbarkeit der ersten Services (5) überprüft. Tritt in diesen Phasen ein Fehler auf, so sind für keinen der Beteiligten Kosten angefallen. Der Fehlerfall wird lediglich durch einen entsprechenden Fehlercode angezeigt (vgl. Kapitel 3.4)

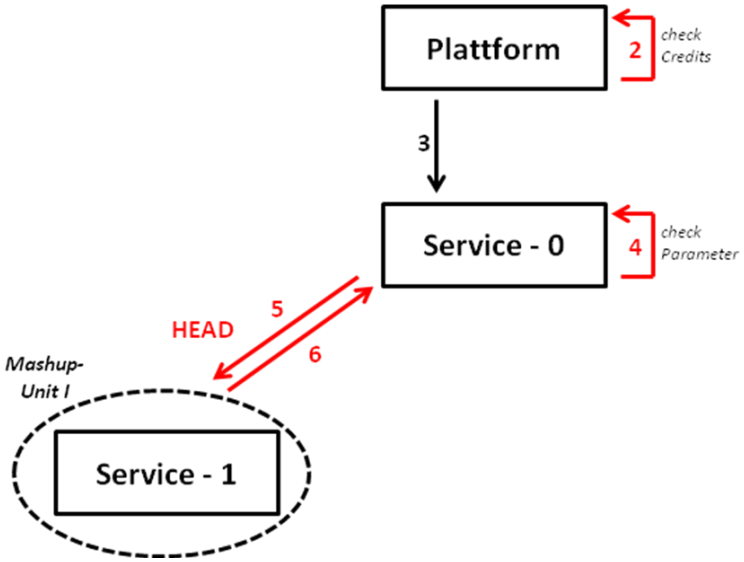

Abbildung 11: Überprüfungen vor Ausführung

3.3.4 Transaktionale Ausführung einer Mashup-Unit

Bei der Diskussion über Transaktionen im Allgemeinen und danach in Bezug auf den REST-HTTP-Kontext ist festgestellt worden, dass die lose Kopplung von Mashup-Anwendungen diesen wiederspricht. Daher soll nachfolgend die Idee der Mashup-Units und ihrer transaktionalen Ausführung vorgestellt werden.
Eine Mashup-Unit bezeichnet eine Menge von Diensten, die zusammen ausgeführt ein Ergebnis erzielen, dass gegenüber dem Mashup-Nutzer abrechnungsfähig ist.
Eine solche Einheit hat unter anderem den Vorteil, dass für sie das Konzept der Transaktionen anwendbar ist, da der Grad der Abhängigkeit bei kleineren „verteilten Systemen“ höher sein kann, als bei einer lose gekoppelten „großen“ Mashup-Anwendung.
Damit stellt sich die Frage, wie eine Transaktion im Fall einer Mashup-Einheit aussieht. Eine Transaktion besteht in dem hier vorliegenden Fall aus dem Ausführen der Services und der entsprechenden Bezahlung. In unserem Interaktionsprotokoll zeigt sich dies zunächst mit der vorläufigen Abbuchung des Preises, dann durch die Probe- und „echte“ Service-Anfrage und letztendlich bei fehlerfreier Ausführung mit der Bestätigung des Zahlungsvorgangs. Im Fehlerfall wird diese Transaktion zurückgefahren, das heißt die Dienste abgebrochen und der Abbuchungsvorgang rückgängig gemacht.
Dies ist aber nicht uneingeschränkt möglich, da gemäß den Annahmen aus Kapitel 1 nicht alle Dienste kostenfrei storniert werden können. Damit ist das transaktionale Konzept hier nur eingeschränkt anwendbar. Jedoch ist es vorstellbar, dass aufgrund des Wettbewerbs zwischen den Service-Anbietern, viele Dienste eine kostenlose Stornierung anbieten werden, da diese in den meisten Fällen lediglich geringe Kosten mit sich bringt.
Folglich ist darzulegen, wie Transaktionen in REST beziehungsweise HTTP implementiert werden sollen. Im vorherigen Kapitel wurde der Ansatz von Pautasso (vgl. Kapitel 2.5.2) vorgestellt. An dessen Anwendung des TCC-Patterns orientiert sich die hier beschriebene Idee.

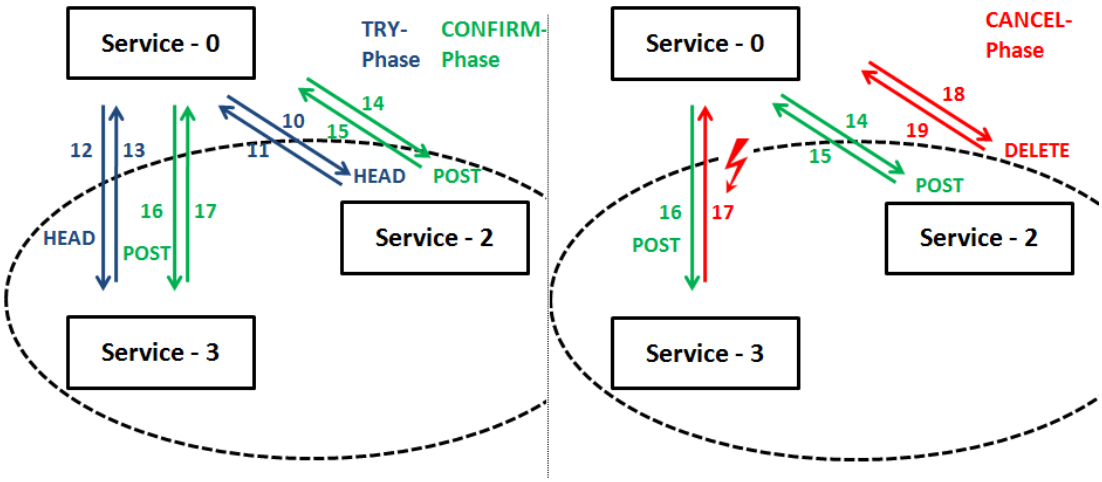

Abbildung 12: Transaktionale Ausführung einer Mashup-Unit

Abbildung 12 beschreibt die Vorgehensweise bei Ausführung einer Mashup-Unit. Diese ist grundsätzlich in zwei verschiedene Phasen aufgeteilt. Zum einen in die TRY-Phase, die aus der vorläufigen Abbuchung und einem HTTP-HEAD-Request besteht und zum anderen in die CONFIRM-Phase, die sich aus der POST-Anfrage, die den Service kostenpflichtig startet und der letztendlichen Bestätigung durch die Plattform zusammensetzt. Der erste HEAD-Request ist kostenlos und soll dazu dienen, Fehlern bei der späteren Ausführung vorzubeugen, in dem er die Verfügbarkeit aller an dieser Transaktion beteiligten Dienste testet. Der Request wirkt sich aber nicht weiter auf den Zustand des entsprechenden Guthabenkontos aus. Dieses wird erst durch die folgende POST-Anfrage verändert. Diese Zustandsänderung ist hier erst einmal gedanklich zu sehen, da erst im späteren Verlauf des Protokolls, genauer bei der Bestätigung der Ausführung, das Kontoguthaben entsprechend angepasst wird. Tritt in dieser zweiten Phase ein Fehler auf, so wird versucht ein Rollback durchzuführen. Dies geschieht durch einen HTTP-DELETE-Request an die bereits ausgeführten Dienste und soll diesen signalisieren, dass sie storniert worden sind und damit weitere Berechnungen abbrechen sollen.

3.4 Behandlung der Fehlerfälle

Nach dem der Ablauf des Protokolls im fehlerfreien Fall vorgestellt worden ist, wird im Folgenden die Behandlung der Fehlerfälle gezeigt.

3.4.1 Nicht ausreichendes Guthaben: fixer Preis

Bei diesem Fehlerfall besitzt der Nutzer kein ausreichendes Guthaben auf seinem Prepaid-Konto, um das Service-Mashup zu starten. Versucht er dies trotzdem, so erzeugt die Plattform eine Fehlermeldung und gibt den HTTP-Status-Code „402 – Payment Required“ zurück. Der Client kann dann unab-

hängig davon sein Konto aufladen und es zu einem späteren Zeitpunkt noch einmal versuchen (vgl. Abbildung 13).

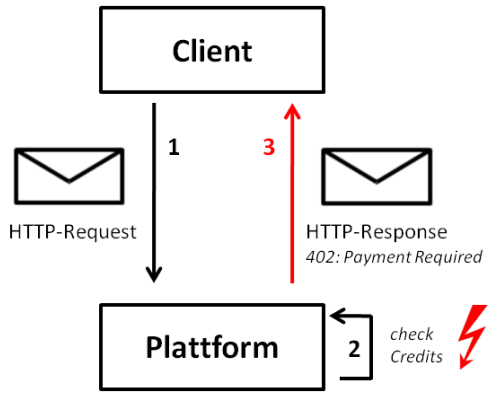

Abbildung 13: Fehlerbehandlung bei nicht ausreichendem Guthaben (fixer Preis)

3.4.2 Nicht ausreichendes Guthaben: variabler Preis

Dieses Fehlerszenario (vgl. Abbildung 14) unterscheidet sich von dem des fixen Preis zu dem Zeitpunkt, in dem der Fehler entdeckt wird. Der Fehler tritt hier erst zur Laufzeit des Mashups auf (17), wenn externe kostenpflichtige Dienste durch den Mashup-Komponisten schon gestartet wurden. Damit die Ausführung nicht mit einer Fehlermeldung abgebrochen werden muss und der Mashup-Anbieter auf den Kosten sitzen bleibt, sieht das Interaktionsprotokoll folgende Vorgehensweise vor. In der HTTP-Response-Nachricht wird dem Nutzer von der Plattform durch den Status-Code „402: Payment Required“ signalisiert, dass sein Guthaben nicht ausreicht. Gleichzeitig wird ihm aber die Möglichkeit angeboten, sein Guthaben aufzuladen oder die Menge der Datensätze einzuschränken. Dies kann zum Beispiel durch einen im Nachrichten-Body mitgeführten Link geschehen, der auf eine entsprechende Webseite zeigt. Dies ist in Abbildung 14 durch den zusätzlichen Payload der HTTP-Response (18) angedeutet. Die Auswahl des Clients (19) wird durch die Plattform überprüft und dann im positiven Fall das Mashup normal weiter ausgeführt (Abbildung 14, linke Hälfte), beziehungsweise wie in der rechten Hälfte angedeutet, die Ausführung der Mashup-Anwendung abgebrochen. Letztere Situation wird dann gemäß der in der Klasse „fehlerhafte Rückgabe“ vorgestellten Verfahren behandelt.

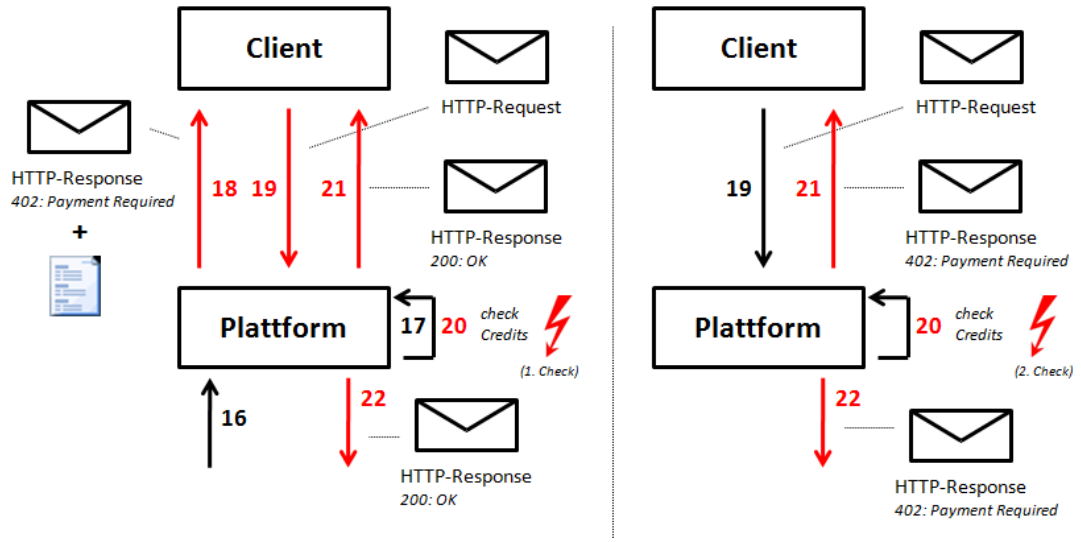

Abbildung 14: Fehlerbehandlung bei nicht ausreichendem Guthaben (variabler Preis)

3.4.3 Falsche Eingabeparameter

In diesem Szenario (vgl. Abbildung 15) startet der Client nach der Prepaid-Bezahlung die Mashup-Anwendung mit Eingabeparametern, die nicht konform zur Spezifikation sind. Die dadurch verursachte fehlerhafte Ausführung soll in diesem Interaktionsprotokoll verhindert werden, in dem der Mashup-Komponist die Input-Daten auf ihre Korrektheit hin überprüft (4). Ist dies nicht der Fall, so wird die Anwendung nicht weiter ausgeführt und dem Nutzer der HTTP-Status-Code „415: Unsupported Media Type“ zurückgegeben.
Die bereits erfolgte vorläufige Abbuchung wird durch das Ausbleiben einer Bestätigung seitens der Plattform wieder rückgängig gemacht. Da der Mashup-Komponist den Fehler vor der kostenpflichtigen Inanspruchnahme der externen Dienste erkennt, treten für ihn in diesem Szenario keine Kosten auf.
Bleibt noch der Fall zu betrachten, in dem die Eingabeparameter gemäß der Spezifikation konform sind, trotzdem aber zu einem Fehler im Ablauf des Mashups führen. Dieses Fehlerszenario fällt unter die Fehlerklasse der „fehlerhaften Rückgabe“ und die Behandlung in diesem Fall wird in Kapitel 3.4.5 erläutert.

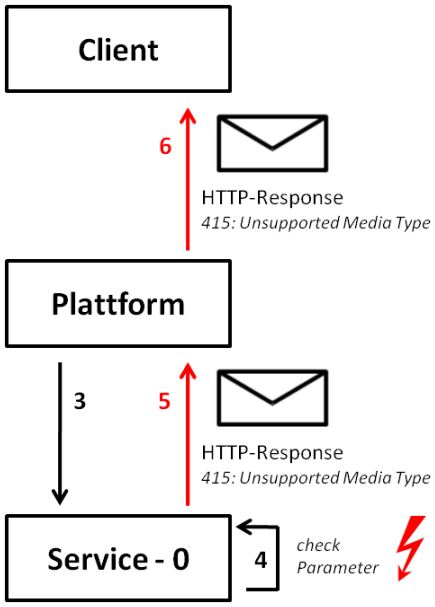

Abbildung 15: Fehlerbehandlung bei falschen Eingabeparametern

3.4.4 Dienst nicht verfügbar

Ein weiteres Problem tritt auf, wenn der Nutzer nach der Bezahlung das Mashup starten will und dieses nicht verfügbar ist. Dabei spielt es keine Rolle, ob der Mashup-Komponist oder ein Service aus der zuerst auszuführenden Einheit nicht verfügbar ist (vgl. Abbildung 16). Das hier vorgestellte Protokoll spezifiziert für diesen Fall folgenden Ablauf. Vor dem eigentlichen Service-Aufruf seitens des Mashup-Komponisten wird mit einer HTTP-HEAD-Anfrage getestet, ob der entsprechende Service verfügbar ist. Diese HTTP-Anfrage enthält keine Eingabeparameter und dient nur als Erreichbarkeitstest. Die zu erwartende Antwort auf diesen Request sollte die Rückgabe des HTTP-Status-Codes „200: OK“ sein. Der Erhalt einer Fehlermeldung wird auf den HTTP-Status-Code „503: Service Unavailable“ abgebildet.
Da dieser Test keine Kosten mit sich bringt, kann im Fehlerfall die vom Client bezahlte Kompensation durch das Unterbleiben der Bestätigung rückgängig gemacht werden.
Ist der Dienst trotz des vorherigen positiven Erreichbarkeitscheck bei Start der Ausführung nicht erreichbar oder handelt es sich um einen nicht erreichbaren Service aus den nachfolgenden Units, so werden diese Fehlerfälle auf die Klasse der fehlerhaften Rückgabe abgebildet.

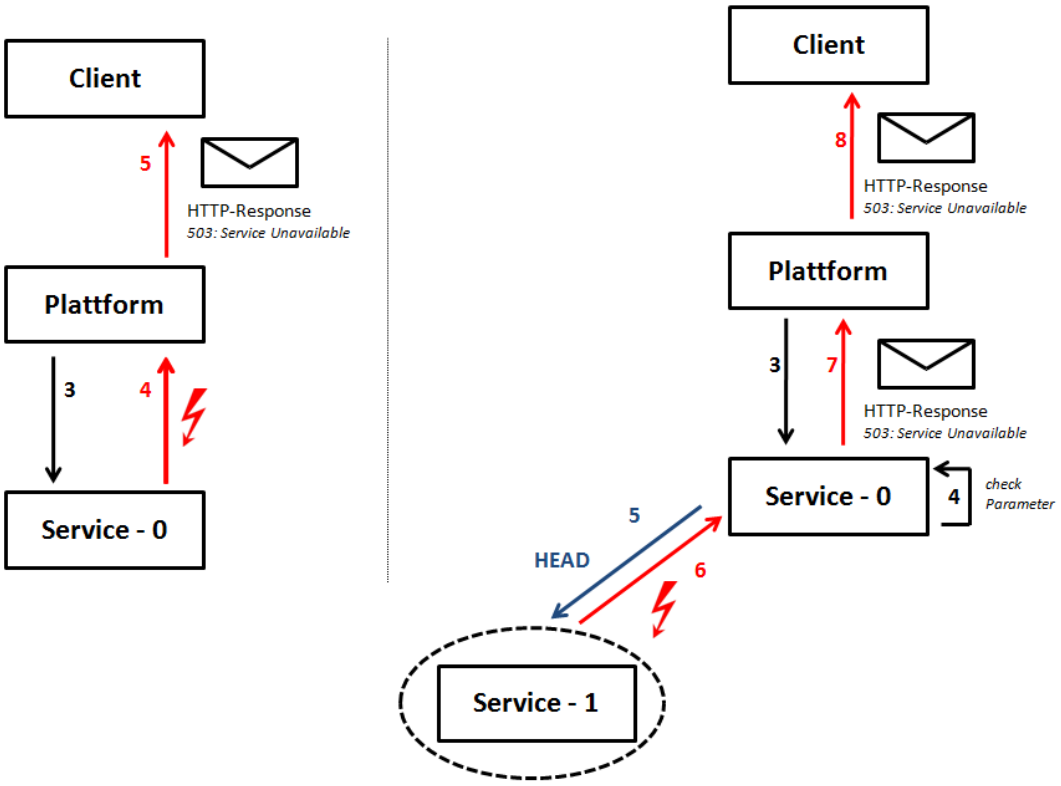

Abbildung 16: Fehlerbehandlung bei einem nicht verfügbaren Dienst

3.4.5 Fehlerhafte Rückgabe

keine Ergebnisse

Bei diesem Fehlerszenario (vgl. Abbildung 17) tritt bei der kostenpflichtigen Ausführung eines Services ein Fehler auf (9), so dass der Mashup-Komponist nicht die angeforderten Daten oder eine Teilmenge davon bekommt. Dies führt zu einem Rollback der Transaktion wie in Abschnitt 3.3.4 beschrieben. Das heißt, er bricht die weitere Ausführung des Mashups ab und storniert bereits ausgeführte Dienste. Als Ergebnis leitet er an die Plattform durch einen HTTP-Callback-Request einen leeren Entity-Body weiter, durch den die Plattform diese Fehlerklasse erkennt. Die Plattform informiert daraufhin den Nutzer mit einem „500: Internal Server Error“, dass bei der Ausführung des Mashups etwas schief gegangen ist. Des Weiteren kompensiert die Plattform diejenigen Services, die nicht kostenfrei storniert werden konnten und bereits ausgeführt worden sind. Hier liegt das Problem dieser Fehlerklasse und auch der kostenpflichten Mashups im Allgemeinen. Durch den Ausfall einzelner externer Komponenten kann es dazu kommen, dass für den Mashup-Anbieter bereits Kosten angefallen sind, er aber keine Kompensation für seinen Dienst bekommt. Daher wird im Folgenden die Idee der transaktionalen Ausführung von Mashups-Units aufgegriffen und ihr Vorteil hinsichtlich des hier vorliegenden Problems erläutert.

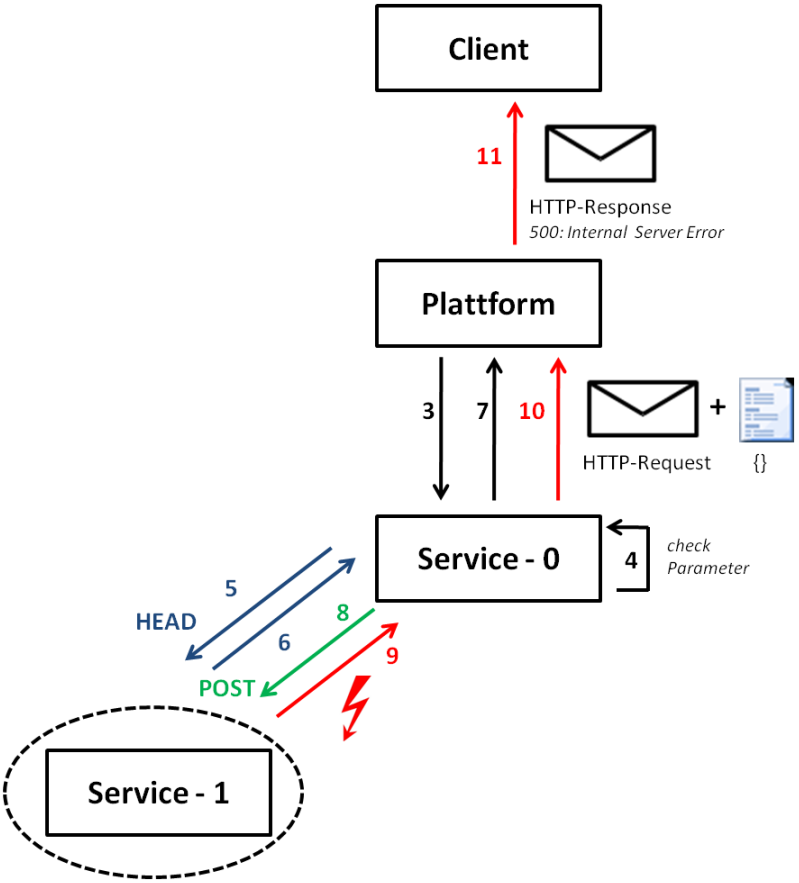

Abbildung 17: Fehlerbehandlung bei fehlerhafter Rückgabe (keine Daten)

teilweise Ergebnisse

Bisher ist nur betrachtet worden, wie mit Hilfe der Units das Transaktionen-Konzept auf das Mashup-Szenario angewendet werden konnte. Dies erlaubte uns zu garantieren, dass in einer Einheit nur bezahlt wird, wenn eine Leistung erfolgt ist. Ansonsten wird die Ausführung abgebrochen und zurückgefahren.
Jedoch bietet dieser Ansatz noch einen weiteren Vorteil. Dadurch, dass diese Teile eines Mashups unabhängig und nacheinander ausgeführt werden, sind bei Auftritt eines Fehlers zum Beispiel gegen Ende der Ausführung durch die vorher ausgeführten Einheiten schon Teilergebnisse produziert worden. Da eine Mashup-Unit gemäß der Definition eine Kombination von Services darstellt, die eine gegenüber dem Nutzer abrechenbare Leistung erzielt, wird der Mashup-Komponist für diese Teilergebnisse kompensiert.
Konkret spezifiziert das Protokoll den in Abbildung 18 gezeigten Ablauf. Dabei werden nach dem Rollback der ausgeführten Transaktion per HTTP-Callback-Request die bereits gesammelten Teilergebnisse an die Plattform weitergeleitet (20), die die entsprechende Kompensation hierfür ausführt (21) und dann diese Daten mit beschreibender Fehlermeldung an den Client weiterleitet (22).

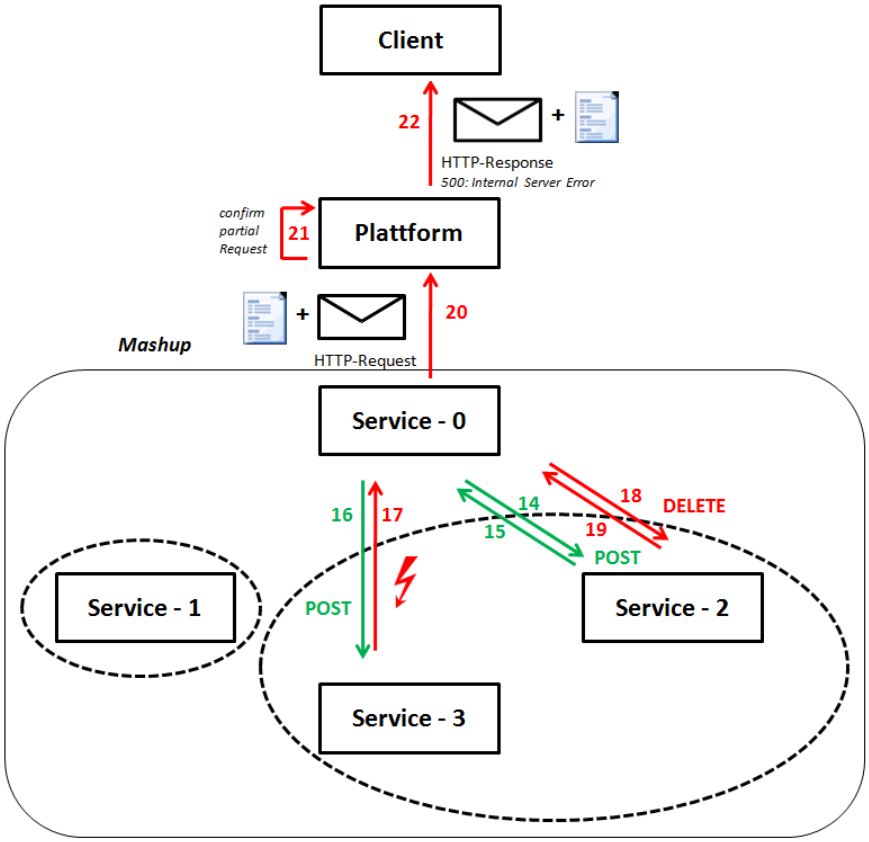

Abbildung 18: Fehlerbehandlung bei fehlerhafter Rückgabe (teilweise Ergebnisse)

4 Implementierung des Prototyps

Nachdem im vorherigen Kapitel das Interaktionsprotokoll spezifiziert worden ist, wird es nun prototypisch im Rahmen des Cocktail-Projektes implementiert. Mit dieser Umsetzung wird die Plausibilität der Spezifikation gezeigt.
Hierfür werden als erstes die Anforderungen für die Umsetzung diskutiert, insbesondere mit Blick auf die konkrete Implementierung des Protokolls.
Danach erfolgt die Beschreibung der zugrunde liegenden Architektur. Diese ist zum einen durch das Internet als Anwendungsplattform für die Mashups und zum andern durch das Cocktail-Projekt festgelegt, in dessen Rahmen die Umsetzung stattfindet.
Im nächsten Abschnitt erfolgt die Implementierung der Spezifikation. Dabei werden die Funktionalitäten des Protokolls auf Klassenebene abgebildet.
Als letztes erfolgt die Beschreibung eines Beispielszenarios, mit dem die Funktionsweise des Protokolls und dessen Umsetzung veranschaulicht und evaluiert wird.

4.1 Anforderungen

Im Folgenden werden die Merkmale skizziert, die für die Implementierung des Interaktionsprotokolls wichtig sind. Dagegen werden die Anforderungen an das System als Ganzes, wie zum Beispiel die Skalierbarkeit oder Verfügbarkeit nicht betrachtet, da die Systemarchitektur durch das Internet und das Cocktail-Projekt vorgegeben ist. Die Anforderungen werden definiert durch die verschiedenen Gruppen, die an dem System beteiligt sind. In unserem Fall lassen sich die Mashup-Nutzer, die Plattform-Betreiber und die Service-/ Mashup-Provider unterscheiden. Ihre unterschiedlichen Anforderungen werden nachfolgend betrachtet.

4.1.1 Funktionale Anforderungen

Spezifikation umsetzen

Als funktionale Anforderung für die Implementierung des Interaktionsprotokolls ist das Umsetzen der Spezifikation zu nennen, die ihrerseits wiederum Kriterien wie die Behandlung der verschiedenen Fehlerfälle erfüllen muss. Diese Anforderung wird von allen Interessengruppen geteilt.

4.1.2 Nicht-funktionale Anforderungen

Performance

Das Kriterium der Performance wurde schon an die Protokollspezifikation an sich gestellt, jedoch besitzt es bei der Implementierung eine andere Dimension. Wurde bei der Beschreibung des Protokolls darauf geachtet, den Ablauf

effizient zu gestalten oder den Payload der Nachrichten so gering wie möglich zu halten, so ist bei der Implementierung vor allem die Auswahl der Technologien wichtig. Interessiert an einer guten Performance sind alle oben genannten Beteiligten.

Portierbarkeit

Ein weiteres wichtiges Merkmal ist die Portierbarkeit der Implementierung. Das bedeutet, dass diese nicht nur im Cocktail-Projekt einsetzbar sein soll, sondern in den meisten Mashup-Szenarios, die eine Plattform für die Abrechnung benutzen. Dies betrifft sowohl die Auswahl der Technologien als auch die konkrete Implementierung. Alle Nutzergruppen, die das Protokoll implementieren müssen, also Plattform-, Mashup- und Service-Anbieter, sind an dieser Anforderung interessiert.

Interoperabilität

Die letzte Anforderung schließt an die der Portierbarkeit an. Damit die hier vorgestellte Lösung einen großen Anwendungsbereich erschließen kann, muss auch die Interoperabilität gegeben sein. Das heißt, dass zum Beispiel externe Dienste ohne großen Aufwand sich in das System integrieren lassen.

4.2 Architektur

In den nächsten Abschnitten wird die Architektur des Mashup-Szenarios vorgestellt und diskutiert, warum diese sich entsprechend darstellt.

4.2.1 Allgemeines

Die grundlegende Architektur in unserem Fall ist durch die Ausführungsumgebung einer Mashup-Anwendung, dem Internet, festgelegt. Dieses stellt abstrahiert ein riesiges verteiltes System dar, dass dem Client/ Server-Prinzip folgt. „Das Client/ Server-Architekturmodell ist ein Systemmodell, in dem das System aus einer Anzahl von Diensten und zugehörigen Servern besteht, sowie aus Clients, die auf diese Dienste zugreifen und sie nutzen.“ [Somm07; S. 283] Weiterhin wird die Architektur durch das Cocktail-Projekt festgelegt, in dessen Kontext die Implementierung erfolgt. Hierbei ist die zentrale Rolle der Plattform zu nennen, die wesentliche Infrastrukturdienste bereitstellt. Wichtig ist in unserem Zusammenhang vor allem die Verwaltung des Prepaid-Systems, auf welches bei einer kommerziellen Mashup-Ausführung zugegriffen werden muss. Diese zentrale Rolle spiegelt sich in Abbildung 19 dahingehend wieder, dass die Kommunikation nicht direkt zwischen Nutzer und Mashup erfolgt, sondern indirekt über die Plattform.

4.2.2 Systemarchitektur

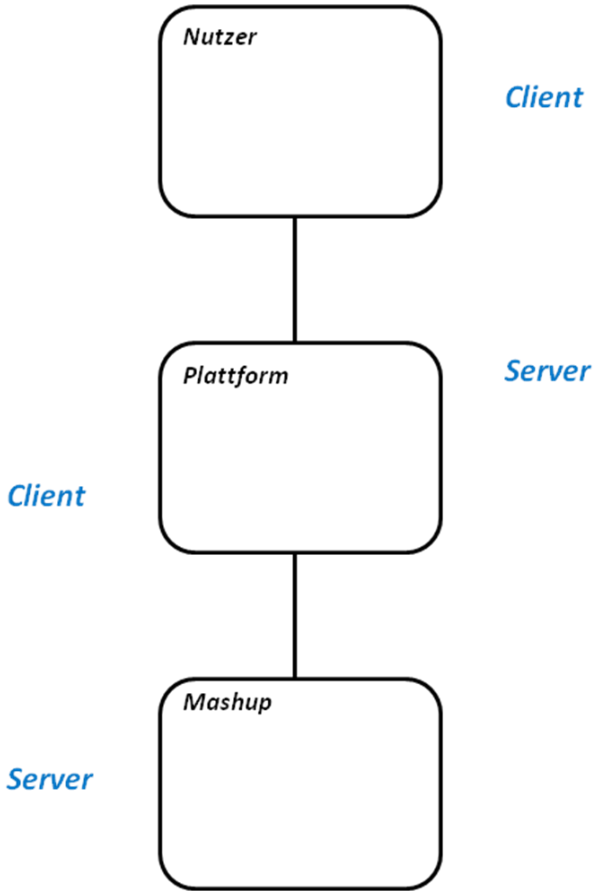

Abbildung 19: Systemarchitektur

Den Aufbau des Systems zeigt Abbildung 19. Dieses besteht aus den Komponenten Nutzer, Plattform und Mashup. Der Nutzer ist der Konsument der Mashup-Anwendung. Die Plattform übernimmt vor allem administrative Aufgaben, wie zum Beispiel die Registrierung von Services. Außerdem ist sie für den Abrechnungsvorgang zuständig. Das Mashup steht für den Service, den der Nutzer gegen ein Entgelt aufrufen kann. Dieses stellt wiederum für sich ein Teilsystem dar (vgl. Abbildung 20).

Beziehungen bestehen zum einen zwischen Nutzer und Plattform und zum anderen zwischen Plattform und Mashup. Hierbei stehen diese jeweils in einem Client/ Server-Verhältnis, das heißt der aktive Client ruft die Dienste des passiven Servers auf. Zu erwähnen ist, dass keine direkte Beziehung zwischen Nutzer und Mashup vorliegt. Dies ermöglicht der Plattform den eingehenden Request direkt hinsichtlich des dafür notwendigen Guthabens zu überprüfen und bei der Auslieferung der Ergebnisse die Bestätigung des Abrechnungsvorganges vorzunehmen.

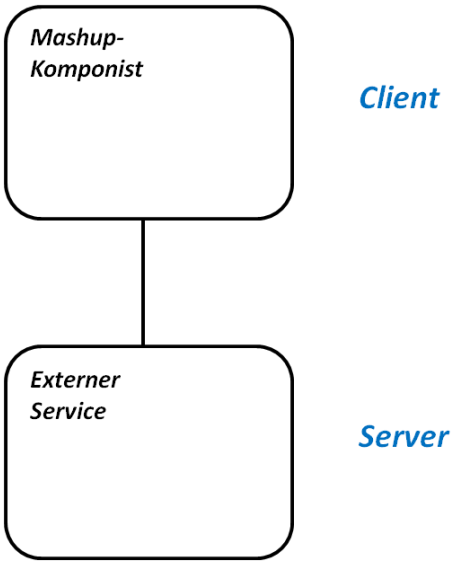

Abbildung 20: Teilsystem Mashup

Die Mashup-Anwendung besteht ihrerseits aus den Komponenten des Mashup-Komponisten und den externen Services. Der Mashup-Komponist ist dabei der Ersteller und Anbieter der Mashup-Anwendung. Die externen Services stellen die verwendeten Ressourcen dar.

Diese zwei Komponenten stehen in einem Client/ Server-Verhältnis, das heißt, der Mashup-Ersteller ruft als Client die auf einem Server liegenden Ressourcen auf.

Eine Beziehung gibt es nur zwischen Mashup-Anbieter und den externen Service. Diese existiert nicht zwischen den verschiedenen Diensten. Das ist darauf zurückzuführen, dass diese von verschiedenen externen Anbietern stammen und damit in den meisten Fällen unabhängig voneinander sind.

4.2.3 Komponenten und Module

Abbildung 21 zeigt die Zuordnung der Module zu den oben vorgestellten Komponenten. Dabei ist zu beachten, dass nur die das Protokoll implementierenden Module zu sehen sind. Nicht dargestellt sind unter anderem Einheiten, die die HTTP-Anfragen entgegennehmen und weiterleiten wie zum Beispiel ein Webserver oder die Implementierung eines REST-Frameworks.

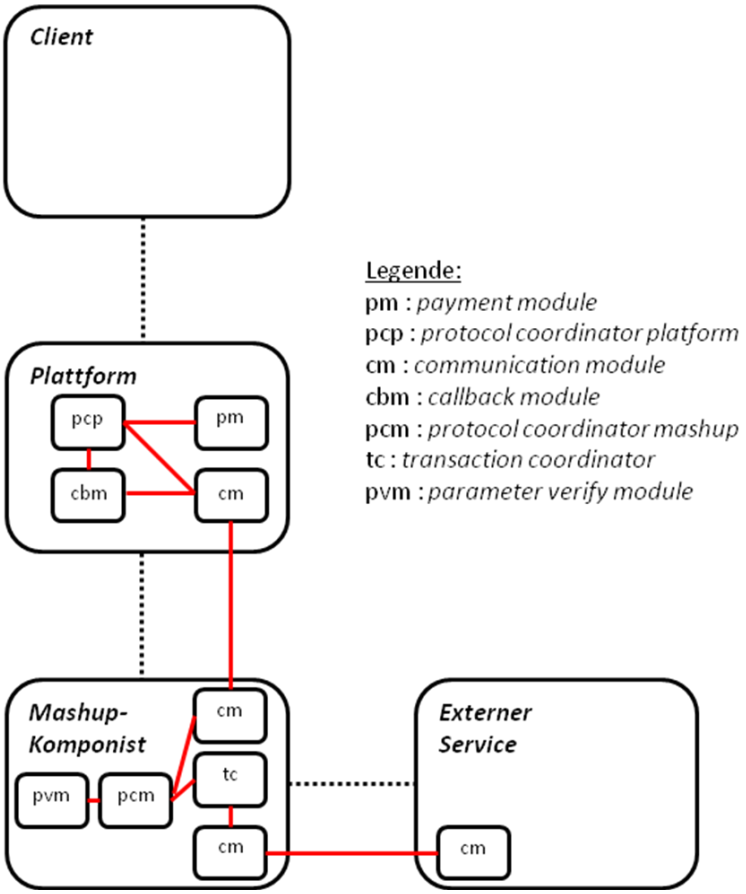

Abbildung 21: Komponenten und Module

Payment-Module

Dieses Modul stellt die Funktionalitäten bereit, die für den Abrechnungsvorgang benötigt werden. Hierzu zählen unter anderem das Überprüfen des Guthabens am Anfang und die Bestätigung des Abbuchungsvorgangs am Ende der Ausführung.
Das „Payment-Module“ wird auf der Plattform ausgeführt und vom „Protocol-Coordinator-Platform“-Modul aufgerufen.

Protocol-Coordinator-Platform

Der Protokoll-Koordinator koordiniert den Ablauf des Interaktionsprotokolls auf der Plattform. Dazu gehören das Einsetzen des „Payment-Module“, sowie des „Communication-Module“. Außerdem delegiert er das Verwalten des Callback-Aufrufs an das „Callback-Module“.
Der Koordinator läuft ebenfalls auf der Plattform und wird nach der Entgegennahme des Mashup-Requests ausgeführt.

Communication-Module

Das Kommunikationsmodul hat die Aufgabe, den für das Interaktionsprotokoll benötigten zuverlässigen Nachrichtenaustausch zu implementieren. Hierfür

werden für beide Enden der Kommunikation die entsprechenden Funktionalitäten bereitgestellt.
Dieses Modul wird einerseits für die Kommunikation zwischen Plattform und Mashup und andererseits zwischen Mashup-Komponist und externem Service genutzt.

Protocol-Coordinator-Mashup

Der Protokoll-Koordinator des Mashup-Komponisten setzt den Ablauf des Interaktionsprotokolls auf der Seite des Mashups um. Zu seinen Aufgabenbereichen gehören unter anderem das Aufrufen des „Parameter-Verify-Module“, sowie das Starten des „Transaktion-Coordinator“. Außerdem nutzt er für das Ausführen des Callback-Aufrufes das „Communication-Module“.
Der Koordinator wird im Mashup-Komponist ausgeführt und wird nach dem Mashup-Aufruf der Plattform durch das Kommunikationsmodul gestartet.

Transaction-Coordinator

Der Transaktions-Koordinator steuert die transaktionale Ausführung einer Mashup-Unit. Hierfür nutzt er die Funktionalitäten seitens des Kommunikationsmoduls für die verschiedenen Aufrufe.
Dieses Modul wird ebenfalls auf dem Mashup-Komponisten ausgeführt und wird durch den Protokoll-Koordinator gestartet.

Parameter-Verify-Module

Dieses Modul soll die Aufgabe erfüllen, eingehende Mashup-Requests hinsichtlich ihrer Eingabeparameter zu überprüfen.
Wie auch der Transaktions-Koordinator wird dieses Modul auf dem Mashup-Komponisten ausgeführt. Aufgerufen wird es durch das Protokoll-Koordinator-Modul des Mashup-Erstellers.

4.3 Umsetzung

Dieser Abschnitt beschreibt die konkrete Implementierung des Interaktionsprotokolls. Hierzu werden im Folgenden zunächst die ausgewählten Technologien und danach die Abbildung der Funktionalitäten auf Klassenebene beschrieben.

4.3.1 Technologien

Java

Als Programmiersprache für die Umsetzung der Protokollspezifikation wurde Java ausgewählt. Dies lässt sich vor allem darauf begründen, dass die Implementierung in das Cocktail-Projekt eingebettet werden soll und dieses hauptsächlich in Java realisiert worden ist. Des Weiteren wirken sich die Vor-

teile von Java wie zum Beispiel die Plattformunabhängigkeit hinsichtlich der Anforderung der Portierbarkeit der Lösung positiv aus.

REST-Framework

Als nächste wichtige Frage ist zu klären, wie eingehende HTTP-Requests von den verschiedenen Komponenten angenommen beziehungsweise abgesetzt werden können.
Die vom Cocktail-Projekt eingesetzte Plattform beinhaltet ein eigenes REST-Framework, das für andere REST-Schnittstellen bereits eingesetzt wird. Des Weiteren soll die hier vorgestellte Implementierung in diese Plattform integriert werden und daher wird dieses REST-Framework im Folgenden verwendet. Für das Deployment der Plattform-Module soll der ebenfalls bei der Plattform eingesetzte Tomcat-Server verwendet werden.
Bei der Implementierung der verschiedenen Services sowie des Mashup-Komponisten kommt eine Kombination aus Jetty-Web-Server für die Annahme von HTTP-Requests und der HTTP-Client-Bibliothek für das Absetzen von HTTP-Anfragen zum Einsatz.

JSON

Als Ausgabeformat wird in dieser Implementierung das Dateiformat JSON benutzt. Auch dies ist eine Vorgabe aus dem Cocktail-Projekt. Der Vorteil von JSON ist seine Kompaktheit, mit der zum Beispiel gegenüber der Verwendung von XML wertvolle Verarbeitungszeit gespart werden kann. Außerdem sind in Java zahlreiche Bibliotheken für die Verarbeitung von JSON vorhanden.

4.3.2 Klassendiagramme

Abbildung 22 zeigt die Klassen, die die Funktionalitäten der Module aus Abbildung 21 implementieren und ihre Beziehungen untereinander. Zentral sind hierbei die beiden Klassen „RequestHandlerPlatform“ und „RequestHandlerMashup“, die den Protokollablauf auf Seiten der Plattform beziehungsweise des Mashup-Anbieters umsetzen. Die Kommunikation zwischen den Beteiligten erfolgt über den „ServiceCommunicationService“, der von allen Partien implementiert wird. Die Services sind hier nur als Interface definiert, damit sie je nach Anwendungsbereich spezifisch implementiert werden können. Die in der Abbildung dargestellten Beziehungen zu den Service-Interfaces beziehen sich eigentlich auf die konkrete Implementierung der Schnittstelle. Dies wurde hier aus Gründen der Übersichtlichkeit vereinfacht dargestellt. Nachfolgend werden die Klassen hinsichtlich ihrer Funktionalitäten und ihrer Zugehörigkeit zu den Modulen kurz beschrieben.

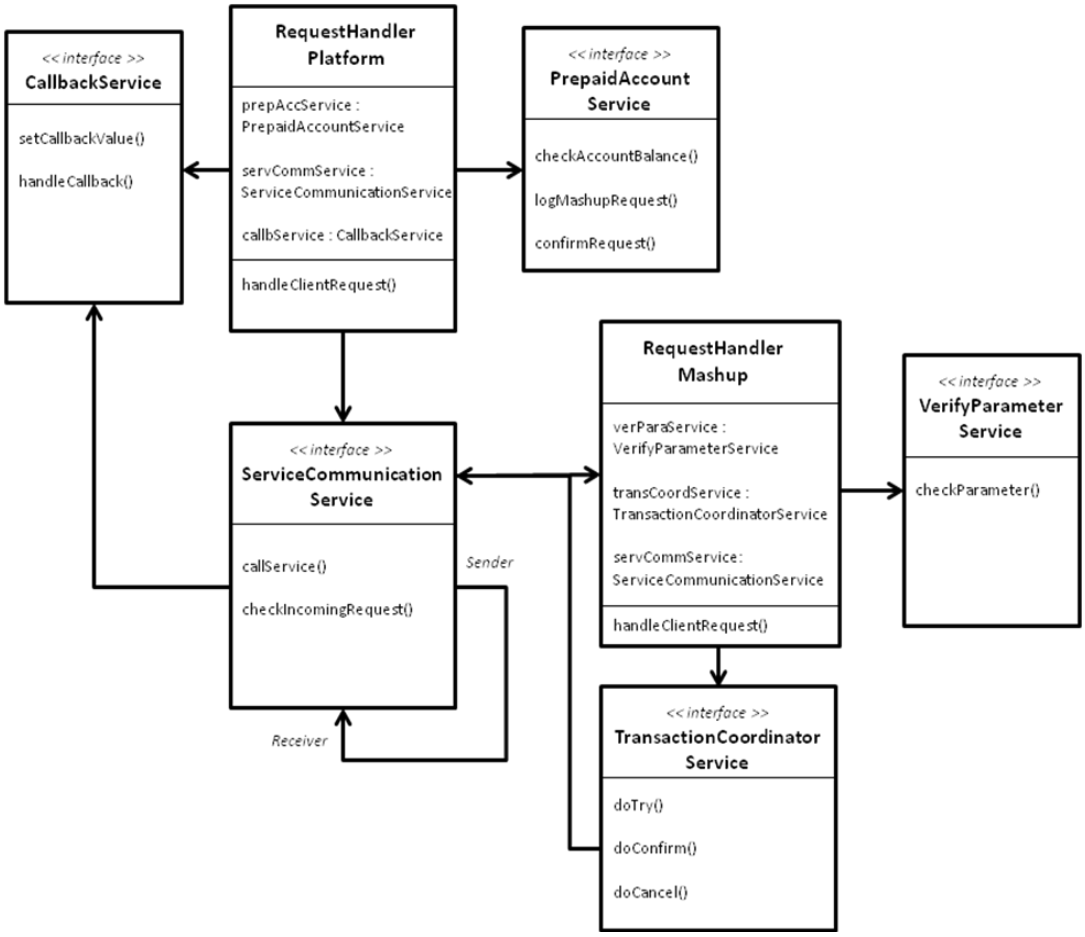

Abbildung 22: Klassendiagramm

Payment-Module

Der „PrepaidAccountService“ bildet den Kern des Abrechnungsmoduls. Dieser stellt über die Methode „checkAccountBalance“ die Funktionalität für das Überprüfen des Guthabens am Anfang des Protokolls bereit. Des Weiteren ist die Methode „logMashupRequest“ für das Protokollieren der verschiedenen Service-Aufrufe zuständig. Außerdem bestätigt der Service über die „confirmRequest“-Methode den Abbuchungsvorgang. Da die Infrastruktur für die Abrechnung bei verschiedenen Plattformen unterschiedlich aussieht, ist der Service nur als Interface definiert, dessen Implementierung an den entsprechenden Anwendungsfall angepasst werden kann.

Protocol-Coordinator-Platform

Dieses Modul besteht aus der „RequestHandlerPlatform“-Klasse. Diese steuert in der Methode „handleClientRequest“ den Ablauf des Interaktionsprotokolls durch das Aufrufen der verschiedenen Services, die als Attribute der Klasse definiert sind.

Communication-Module

Das Kommunikationsmodul beinhaltet den „ServiceCommunicationService“. Dieser spezifiziert Methoden für die Umsetzung der zuverlässigen Kommunikation. Dabei dient die „callService“-Methode für das Client-seitige Absetzen eines zuverlässigen Requests und die „checkIncomingRequest“-Methode für die Server-seitige Überprüfung einer eingehenden Anfrage. Dieser Service ist ebenfalls nur als Interface definiert, da unterschiedliche Mechanismen für das Garantieren einer zuverlässigen Kommunikation denkbar sind.

Callback-Module

Dieses Modul besteht ebenfalls aus einem Service. Dieser definiert im vorliegenden Fall zwei Methoden für den Umgang mit einem Callback. Zum einen speichert „setCallbackValue“ die durch den Callback übertragenen Daten. Und zum anderen bietet „handleCallback“ den Zugriff auf diese. Wie auch schon bei den obigen Services, ist hier nur das Interface spezifiziert. Die konkrete Implementierung kann dadurch an bereits vorhandene Infrastruktur angepasst werden.

Protocol-Coordinator-Mashup

Die „RequestHandlerMashup“-Klasse steuert wie auch schon der RequestHandler der Plattform den Ablauf des Interaktionsprotokolls, hier aber auf Seiten der Mashup-Anwendung. Dazu nutzt sie die entsprechenden Funktionalitäten der Services, die als ihre Attribute definiert sind. Des Weiteren enthält sie anwendungsspezifischen Code für die Ausführung des Mashups.

Transaction-Coordinator

Der Transaktionen-Koordinator besteht aus einem Service, der die Funktionalität der transaktionalen Ausführung einer Mashup-Unit kapselt. Hierzu bietet er die drei Funktionen „doTry“, „doConfirm“ und „doCancel“ an, die entsprechend ihrem Namen die in dem Protokoll spezifizierten Phasen implementieren (vgl. Kapitel 3.3.4).

Parameter-Verify-Module

Der „VerifyParameterService“ stellt lediglich eine Methode zur Überprüfung der Eingabeparameter bereit. Diese kann abhängig von den Eigenschaften der Mashup-Anwendung unterschiedlich ausfallen. Daher ist dieser Service auch nur als Interface spezifiziert.

4.4 Evaluation

Nachfolgend wird nun ein Beispielszenario vorgestellt, an dem die Plausibilität der hier angeführten Lösung gezeigt wird. Dazu wird im zweiten Abschnitt der Protokollablauf in der Implementierung gezeigt.

4.4.1 Beispielszenario

Für die Evaluation der Protokollspezifikation wird nun ein Beispielszenario vorgestellt. Dieses setzt die in Abbildung 6 skizzierte Situation einer Mashup-Anwendung mit drei externen Diensten um.
Dieses Mashup bietet dem Nutzer an, für potenzielle Kunden ein Kredit-Scoring durchzuführen. Hierfür liest der Mashup-Komponist als erstes aus einer von Service-1 bereitgestellten CRM-Datenbank die Datensätze des Nutzers aus. Danach sendet er diese an Service-2, der die Daten nach potenziellen Interessenten filtert und als letztes berechnet Service-3 den Scoring Wert für die gefilterten Datensätze. Dieses Ergebnis gibt der Mashup-Komponist an den Nutzer zurück. Alle hier aufgerufenen Dienste haben dabei einen fixen Preis.

4.4.2 Ablauf

Wie für die Umsetzung dieses Szenario der konkrete Protokollablauf aussieht, wird in den nachfolgenden Sequenzdiagrammen veranschaulicht. Diese zeigen den Kontrollfluss auf Klassenebene mit den entsprechenden Methodenaufrufen. Die Zugehörigkeit zu den verschiedenen Komponenten aus Abbildung 21 ist durch die farbige Hinterlegung angedeutet. Die Anmerkung „Request delivered by REST-Framework“ soll aussagen, dass der HTTP-Request zuerst über das entsprechende REST-Framework beziehungsweise über ein Servlet an das Modul beziehungsweise an die entsprechende Klasse ausgeliefert wurde.

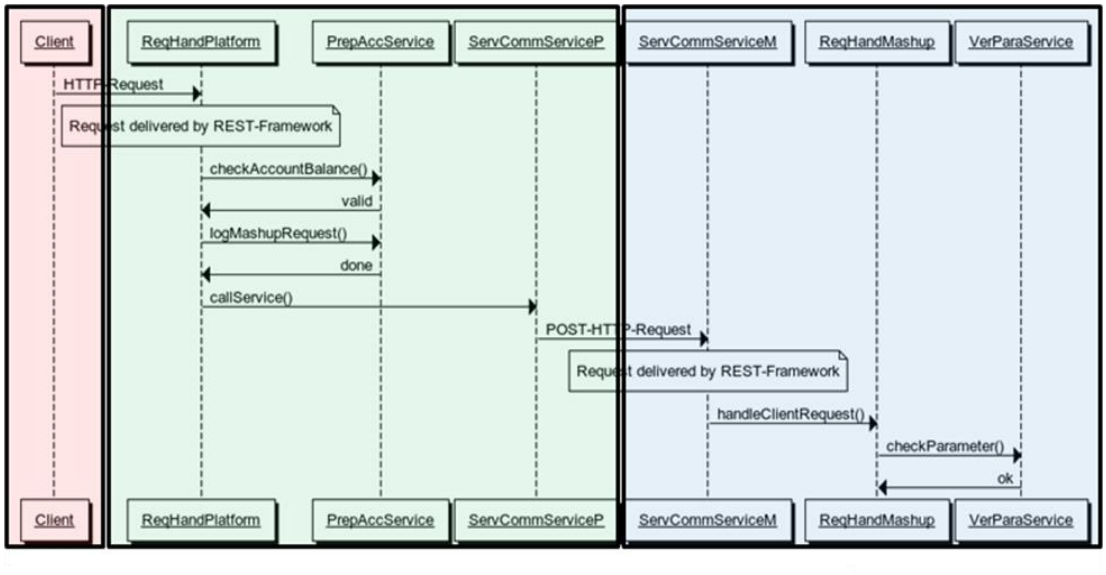

Abbildung 23: Sequenzdiagramm Teil-1

In Abbildung 23 startet der Client (rot hinterlegt) mit einem HTTP-Request aus dem Browser heraus das Mashup. Die „RequestHandlerPlatform“-Klasse (ReqHandPlatform) überprüft dabei zunächst das Guthaben des Nutzers durch Aufruf der entsprechenden Methode des „PrepaidAccountService“ (PrepAccService). Ein weiterer Methodenaufruf loggt die für die Abrechnung relevanten Datenbankeinträge. Nachdem die Ausführung durch den „PrepaidAccountService“ bestätigt worden ist, wird die „callService“-Methode des „ServiceCommunicationService“ (ServCommServiceP) aufgerufen, wodurch ein HTTP-Request an den Mashup-Komponisten (hier blau hinterlegt) gesendet wird. Dessen „ServiceCommunicationService“ (ServCommServiceM), leitet den Request an die „RequestHandlerMashup“-Klasse (ReqHandMashup) weiter. Diese überprüft zunächst durch Aufrufen der „checkParameter“-Methode des „VerifyParameterService“ die Korrektheit der Input-Daten.

Danach wird die erste Mashup-Unit, hier bestehend aus Service-1, aufgerufen (vgl. Abbildung 24). Dafür wird der Transaction-Coordinator mit der Methode „doTry“ gestartet. Diese sendet, ausgeführt durch den „ServiceCommunicationService“, einen HEAD-HTTP-Request an Service-1 (lila hinterlegt), genauer an dessen Kommunikationsmodul (ServCommServiceS1). Nachdem diese Anfrage erfolgreich bestätigt wurde, leitet der „TransactionCoordinatorService“ (TransCoordService) die Response an den RequestHandlerMashup weiter, der die erfolgreiche Erreichbarkeitsüberprüfung der ersten Mashup-Unit gegenüber der Plattform bestätigt. Außerdem startet er durch Senden einer POST-HTTP-Anfrage die Ausführung von Service-1.

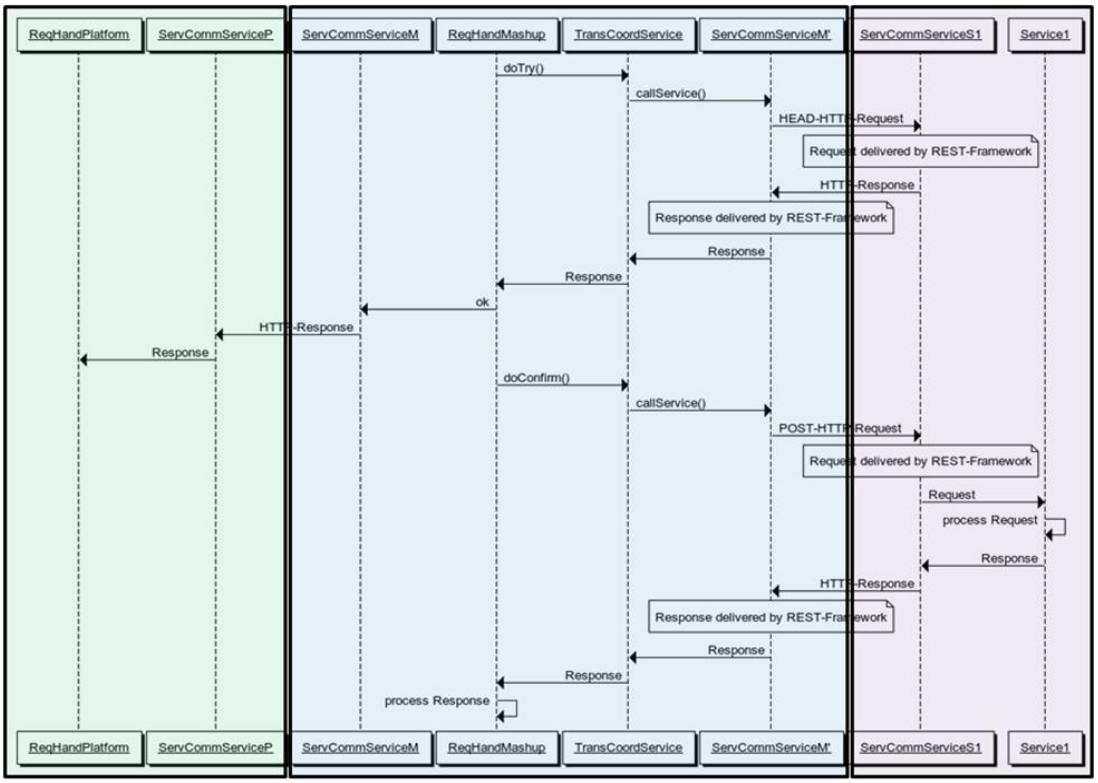

Abbildung 24: Sequenzdiagramm Teil-2

Nachdem die erste Mashup-Unit erfolgreich ausgeführt wurde und damit die Datensätze aus dem CRM-System zurückgegeben wurden, zeigt Abbildung 25 die TRY-Phase der zweiten Einheit. Diese folgt dem Ablauf der ersten Unit, mit dem Unterschied, dass Einheit-2 aus zwei Diensten besteht. Somit erfolgt der HEAD-HTTP-Request zuerst an beide Services, bevor die Ausführung durch die Confirm-Phase (vgl. Abbildung 26, 27) beginnt. Da die Klassenaufrufe ähnlich zu denen in Abbildung 24 sind, wird im Folgenden auf eine genauere Beschreibung verzichtet.

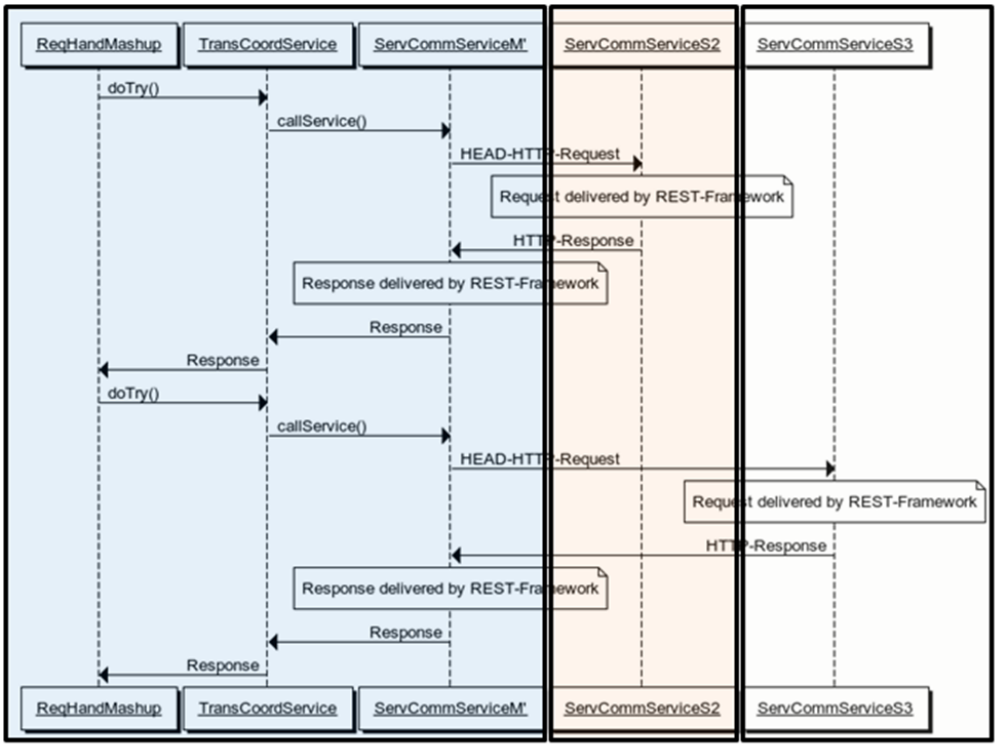

Abbildung 25: Sequenzdiagramm Teil-3

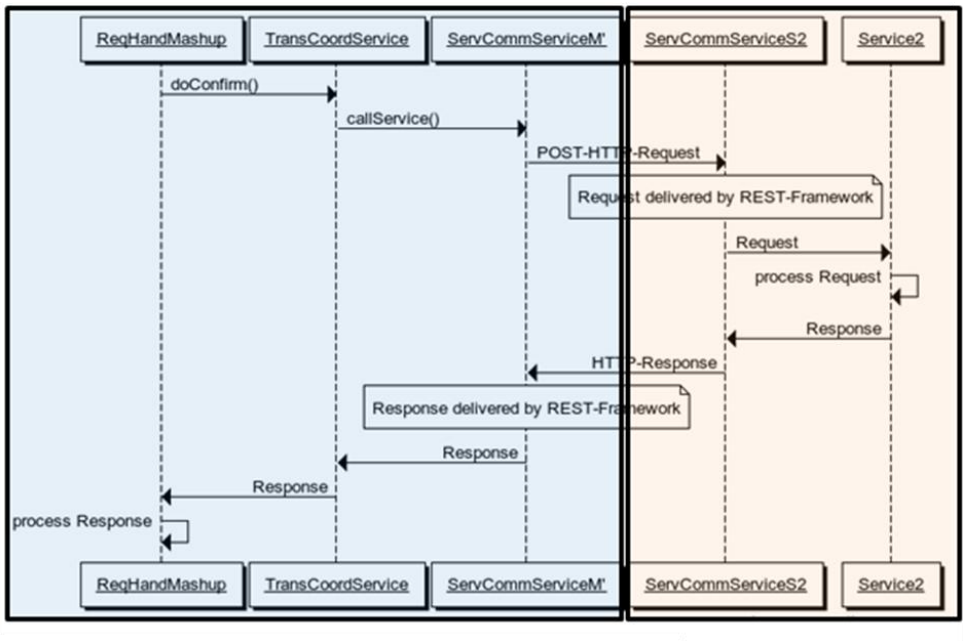

Abbildung 26: Sequenzdiagramm Teil-4

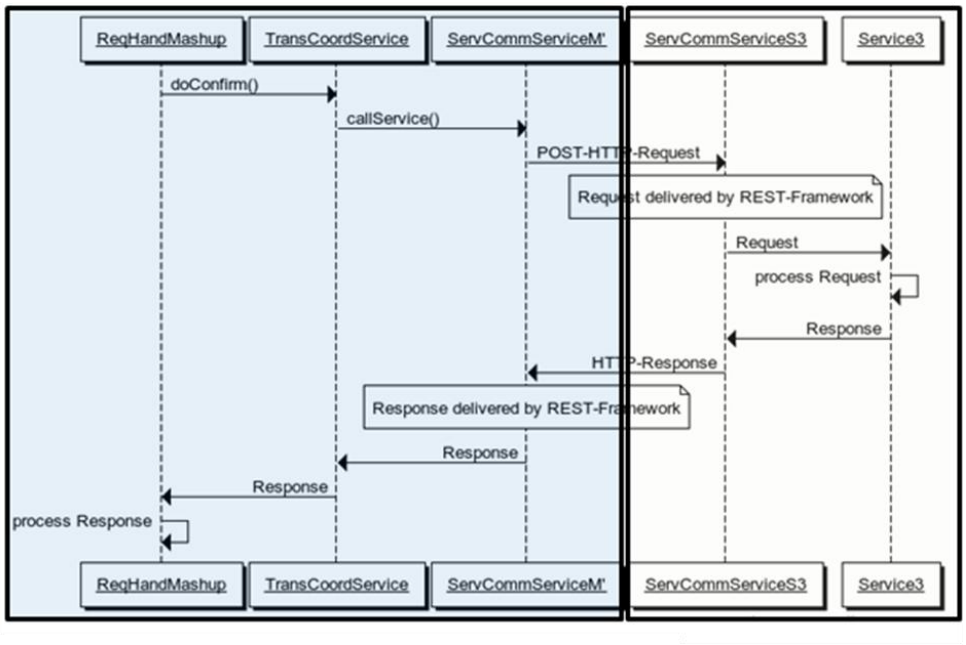

Abbildung 27: Sequenzdiagramm Teil-5

Ist die Ausführung beider Mashup-Units erfolgreich verlaufen und sind damit die Kredit-Scoring Werte für die Interessenten errechnet, so ruft der „RequestHandlerMashup“ über einen Callback die Plattform auf und übergibt ihr die Ergebnisse (vgl. Abbildung 28). Diese bestätigt nach dem Erhalt der Nachricht die erfolgreiche Ausführung des Mashups durch Aufruf der „confirmRequest“-Methode des „PrepaidAccountServices“ und leitet die Daten in der HTTP-Response an den Client weiter.
Problematisch in unserem Szenario ist die lange Wartezeit, die der Client nach dem Senden der HTTP-Anfrage bis zum Eintreffen der HTTP-Response überbrücken muss, da eine HTTP-Connection nicht so lange aufrechterhalten werden kann. Als Lösung wäre denkbar, dass der Client zunächst nach Absenden der Anfrage auf eine neue Webseite weitergeleitet wird und diese dann mit Hilfe von AJAX asynchron nachgeladen wird, wenn die Ergebnisse eingetroffen sind.

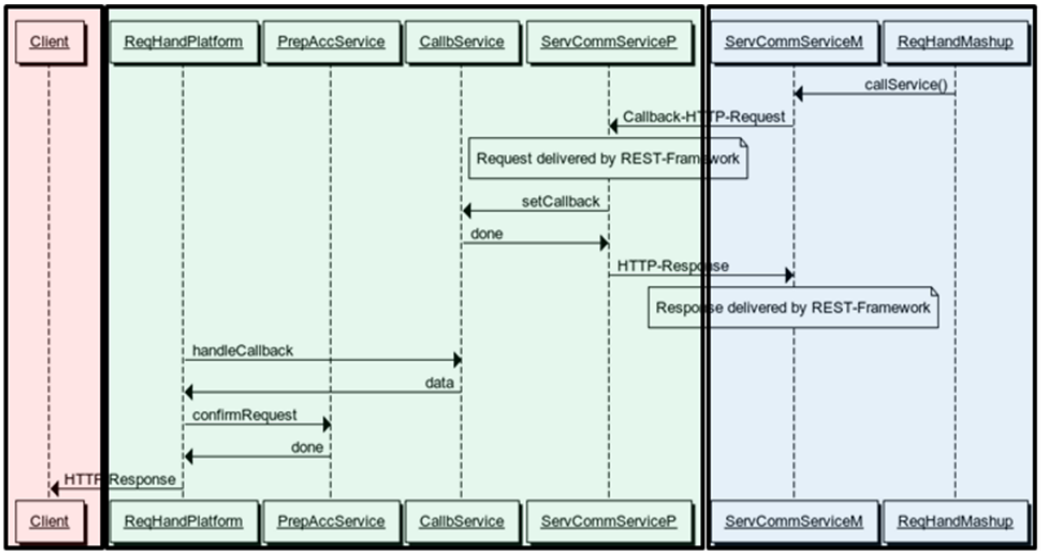

Abbildung 28: Sequenzdiagramm Teil-6

5 Zusammenfassung und Bewertung

In diesem abschließenden Kapitel erfolgt zunächst ein zusammenfassender Überblick über das in dieser Arbeit geleistete. Danach werden das hier vorgestellte Interaktionsprotokoll mit seiner Implementierung diskutiert und Vor- und Nachteile erörtert. Des Weiteren erfolgt ein Ausblick auf Themenbereiche, die in diesem Zusammenhang noch vertieft behandelt werden könnten.

5.1 Überblick

Die Ausgangssituation für diese Arbeit bilden kommerzielle Service-Mashups, die mit einem Prepaid-Preissystem betrieben werden sollen. Die sich daraus ergebenden Schwierigkeiten lassen sich vor allem auf das „Lieferung ⇔ Bezahlung“-Problem abbilden. Dieses beschreibt die Situation, in der eine an dem Geschäft beteiligten Parteien die entsprechende Gegenleistung, Ware oder Geld, nicht bekommt. Des Weiteren existiert im Mashup-Kontext der Sonderfall, dass ein Mashup-Anbieter auf seinen Kosten sitzen bleiben kann, falls ein Fehler die korrekte Lieferung verhindert. Diesen Problemen begegnet das in dieser Arbeit vorgestellte Interaktionsprotokoll einerseits durch die Vermeidung von Fehlerfällen vor der kostenpflichtigen Ausführung und andererseits durch die Anwendung von Transaktionen auf das Mashup-Szenario zur Behandlung von Fehlerfällen während der Ausführung. Letztere Idee zeigt sich in der Form von Mashup-Units, die jeweils einen Zusammenschluss von gegenüber dem Nutzer abrechenbaren Diensten darstellen. Der Vorteil dieser atomaren Einheiten ist, dass sie jeweils für sich transaktional ausgeführt werden können und damit im Fehlerfall ein Rollback anbieten.

5.2 Diskussion

Im Folgenden wird nun das Ergebnis dieser Arbeit diskutiert. Dabei erfolgt zunächst eine Betrachtung der Protokollspezifikation und danach die Diskussion und Bewertung der Implementierung.

5.2.1 Protokollspezifikation

Zunächst wird die Protokollspezifikation diskutiert. Dieser liegen zwei verschiedene Aspekte zugrunde. Zum einen wird der Ablauf der Ausführung einer Mashup-Anwendung festgelegt und zum anderen werden verwendete Technologien wie REST und HTTP ausgewählt und angewendet. Dies zeigt sich vor allem durch die Spezifikation der HTTP-Methoden-Aufrufe und der HTTP-Status-Codes. Der Ablauf des Protokolls ist vor allem durch die grundlegenden Konzepte und die Fehlerbehandlung festgelegt (vgl. Kapitel 3.2 -

3.4). Im Folgenden wird erörtert, inwiefern die an das Protokoll gestellten Anforderungen aus Kapitel 3.1 erfüllt worden sind.

Die wichtigste Aufgabe des Interaktionsprotokolls ist die Behandlung der Fehlerfälle vor und nach der Bezahlung. Wie in Kapitel 3.4 vorgestellt, ist diese erfolgreich durchgeführt worden bis auf den Fall, in dem ein Service nicht kostenlos zurückgefahren werden konnte. Dies ist eine Schwachstelle des Protokolls, die aber durch eine reine Spezifikation des Ablaufs nicht behoben werden kann. In diesem Fall müssen noch andere Mechanismen eingesetzt werden. Denkbar sind zum Beispiel Service-Level-Agreements, die auf vertraglicher Ebene Lösungen für diesen Fall definieren könnten.
Hinsichtlich der Performance sind bezüglich des Ablaufes zwei wesentliche Punkte zu nennen. Der erste Punkt beinhaltet die im Protokoll enthaltenen expliziten Überprüfungen des Guthabens, der Eingabeparameter und der Erreichbarkeit. Diese wirken sich nachteilig auf die Geschwindigkeit der Mashup-Ausführung aus, sind aber notwendige Prozessschritte, wenn man ein Prepaid-Preissystem betrachtet und wenn man dem Fehlerfall der falschen Input-Parameter beziehungsweise des nicht verfügbaren Dienstes vorbeugen will. Der zweite wesentliche Punkt bezieht sich auf die transaktionale Ausführung einer Mashup-Unit. Der hier vorgestellte Ablauf, der sich an Pautasso anlehnt, wirkt sich ebenfalls nachteilig auf die Performance aus. Gegenüber der Verwendung eines „kompletten“ Two-Phase-Commit-Protokolls ist der hier verwendete Ansatz aber schneller.
Hinsichtlich der Auswirkungen der gewählten Technologien auf die Performance ist zu sagen, dass es allgemein ein Vorteil von REST-basierten Services ist, dass sie performant in der Ausführung sind. In der Protokollspezifikation wurde zudem versucht, auf zusätzliche HTTP-Header, deren Verarbeitung ebenfalls Zeit kostet, zu verzichten. Mit einer Ausnahme ist dies gelungen: Für die Umsetzung der Zuverlässigkeit von HTTP mussten eigene Header definiert werden. Da die Zuverlässigkeit des Nachrichtenaustausches die Grundlage der Spezifikation bildet, ist eine dadurch verursachte geringe Geschwindigkeitseinbuße akzeptabel. Es bleibt festzuhalten, dass der Protokollablauf zwar Performance-Einbußen mit sich bringt, dadurch aber zusätzliche Garantien ermöglicht.
Als nächstes wird die Auswirkung der Spezifikation auf den Anwendungsbereich betrachtet. Dies betrifft vor allem die Wahl der Technologien und damit die spezifische Verwendung von HTTP. Mit Ausnahme der Nutzung von eigenen Headern bei der zuverlässigen Kommunikation kommen nur Methoden, Header und Status-Codes zum Einsatz, die in der offiziellen HTTP-Spezifikation definiert sind [vgl. FGMF99]. Dadurch ist ein großer Anwendungsbereich möglich.

5.2.2 Protokollimplementierung

Als nächstes wird die Implementierung der Protokolls diskutiert. Dabei sind ebenfalls zwei Aspekte zu unterscheiden. Zum einen die Auswahl der ver-

wendeten Technologien und zum anderen die konkrete Ausprägung der Implementierung in den verschiedenen Modulen und Klassen.

Als wichtigste Anforderung soll die Implementierung die Spezifikation des Protokolls umsetzen. Vergleicht man die in Kapitel 4.4.2 gezeigten Sequenzdiagramme für die Ausführung des Beispielszenarios mit der Basis-Spezifikation des Protokolls aus Kapitel 3.2., so ist diese Anforderung als erfüllt anzusehen.
Ebenso wie die Spezifikation soll auch die Implementierung performant sein. Hinsichtlich der Technologiewahl sind durch die Wahl von REST und HTTP seitens der Spezifikation die grundlegenden Entscheidungen bereits getroffen worden. Ebenfalls grundlegend ist die Wahl der Programmiersprache. In dieser Implementierung wird Java verwendet. Diese leistungsfähige Programmiersprache wird der Anforderung an eine hohe Prozessgeschwindigkeit gerecht. Des Weiteren wird für das Ausgabeformat JSON verwendet, dessen Kompaktheit ebenfalls von Vorteil für die Geschwindigkeit ist. Ein weiterer Vorteil ist die Definition von Interfaces anstatt von konkreten Klassen, die je nach Anwendungsgebiet spezifisch implementiert und demnach optimal an die jeweilige Infrastruktur angepasst werden können. Nachteilig auf die Geschwindigkeit wirkt sich dagegen der modulare Aufbau der Implementierung aus.
Die Forderung, dass das Protokoll für einen großen Anwendungsbereich gelten soll, wird durch die Portierbarkeit und die Interoperabilität der Implementierung erfüllt. Damit diese in vielen Mashup-Szenarios verwendet werden kann, werden einerseits frei verfügbare Technologien verwendet, wie zum Beispiel Java mit seinen zahlreichen frei verfügbaren Klassenbibliotheken, und andererseits werden die Funktionalitäten in Modulen gekapselt. Dies hat den Vorteil, dass diese ausgetauscht werden können, wie zum Beispiel das Kommunikationsmodul für die Zuverlässigkeit von HTTP. Des Weiteren wirkt sich die Definition von Interfaces an vielen Stellen ebenfalls positiv auf die Portierbarkeit aus, da je nach Anwendungsfall diese spezifisch implementiert werden können.
Ebenso wie die Portierbarkeit ist auch die Interoperabilität der Implementierung wichtig für einen großen Anwendungsbereich, da externe Service-Provider ihre Dienste bei einem zu hohen Integrationsaufwand nicht bereitstellen werden. Dieser wird hier dadurch vermieden, dass der größte Aufwand beim Mashup-Provider liegt, der aber auch den größten Nutzen aus der Implementierung des Protokolls bezieht. Die externen Dienste müssen lediglich die Komponente für die zuverlässige Kommunikation einbinden.

5.2.3 Fazit

Es bleibt festzuhalten, dass die hier vorgestellten grundlegenden Probleme bei einer kostenpflichtigen Mashup-Anwendung, nämlich die Lieferung-Bezahlung-Problematik aus Sicht des Nutzers und des Mashup-Erstellers, durch das hier vorgestellte Interaktionsprotokoll und seiner Implementierung

adressiert und fast vollständig behandelt werden. Um einen sicheren Prozessablauf zu garantieren, müssen aber Performance-Nachteile in Kauf genommen werden.

5.3 Ausblick

Als Abschluss dieser Arbeit werden angrenzende Themenbereiche beziehungsweise Fragestellungen aufgezeigt, die noch weiter untersucht werden könnten.

Als erstes wird diskutiert, inwieweit der in dieser Arbeit verwendete Ansatz auf andere Preissysteme angewendet werden kann. In der Praxis kommt im Mikro-Payment-Bereich häufig auch ein Inkasso-System zum Einsatz, das heißt, dass Beträge beim Verwalter des Abrechnungssystems zwischengespeichert und erst gegen Ende des Monats in einer Sammelrechnung gegenüber dem Kunden abgerechnet werden. Würde ein solches Preissystem auf die in dieser Arbeit dargestellte Problematik übertragen, so wäre das Interaktionsprotokoll auch hierfür weitestgehend anwendbar, jedoch mit kleinen Änderungen. Zum Beispiel würde die Überprüfung des Guthabens vor Ausführung der Mashup-Anwendung wegfallen und damit die darauf bezogenen Fehlerfälle. Die Mechanismen für die Behandlung der anderen Fehlerszenarien, insbesondere das Konzept der Mashup-Units, würden aber ihre Gültigkeit behalten.

Weiterhin interessant ist das Thema HTTP und Zuverlässigkeit, da der hier gewählte Ansatz den Nachteil hat, dass zusätzliche Header definiert werden müssen und damit die Interoperabilität eingeschränkt ist. Hier wäre ein Konzept wünschenswert, das ohne diese individuelle Anpassung des HTTP-Protokolls auskommt.

Abschließend noch eine Anmerkung zum Mashup-Unit-Konzept. Dieses wurde hier nur im Rahmen eines Interaktionsprotokolls umgesetzt. Denkbar ist auch, dass durch Service-Level-Agreements der Ansatz verfeinert wird. Insbesondere die Art der Vergütung einer solchen Einheit sollte auf vertraglicher Ebene geregelt werden. Außerdem kann auch explizit festgelegt werden, in welchen Fällen eine entsprechend ausgestaltete Kompensation erfolgt.

Anhang A

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
CRM	Customer-Relationship-Management bezeichnet die Dokumentation und Verwaltung von Kundenbeziehungen
API	Ein Application Programming Interface beschreibt einen Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.
ACID	ACID ist ein Akronym aus der Informatik. Dabei stehen die einzelnen Buchstaben für die folgenden Eigenschaften einer Transaktion: atomicity, consistency, isolation und durability.

Literaturverzeichnis

[BeDS08] Benslimane, B.; Dustdar, S.; Sheth, A.: *Services Mashups: The New Generation of Web Applications, Internet Computing, IEEE*, vol. 12, S. 13-15, Okt. 2008.

[CCHZ08] Carl, D.; Clausen, J.; Hassler, M.; Zund, A.: *Mashups programmieren - Grundlagen, Konzepte, Beispiele Mashup your life!*, 1. Auflage, O'Reilly, 2008.

[Cock11] ---, *Cocktail : skalierbare, KMU-zentrierte Mashup & SaaS Diensteplattform - Projektidee*. http://www.cocktail-projekt.de/index.php?option=com_content&task=view&id=13&Itemid=33, Abruf am 13.04.2011.

[DaUl04] Dannenberg, M.; Ulrich, A.: *E-Payment und E-Billing: Elektronische Bezahlsysteme für Mobilfunk und Internet*, 1. Auflage, Gabler, 2004.

[FGMF99] Fielding, R.T.; Gettys, J.;Mogul, J.C.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T.: *Hypertext Transfer Protocol -- HTTP/1.1*, W3C, RFC 2616, 1999, http://tools.ietf.org/pdf/rfc2616.pdf.

[Field00] Fielding, R.T.: *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. Thesis, University of California, Irvine, California, 2000.

[Gola05] Goland, Y.: *SOA-Reliability (SOA-Rity) for HTTP draft-goland-http-reliability-00*, 2005, http://www.goland.org/draft-goland-http-reliability-00.text.

[GrRe93] Gray, J.; Reuter, A.: *Transaction Processing. Concepts and Techniques*, Morgan Kaufmann, 1993.

[Litt09] Little, M.: *REST and transaction, 2009,* http://www.infoq.com/news/2009/06/rest-ts.

[MRMK09] Marinos, A.; Razavi, A.; Moschoyiannis, S.; Krause, P.: RETRO: A Consistent and Recoverable RESTful Transaction Model, in *Proc. of the IEEE International Conference on Web Services (ICWS 2009), pages 181-188,* Los Angeles, CA, USA, July 2009.

[Nott05] Nottingham, M.: *POST Once Exactly (POE) draft-nottingham-http-poe-00,* 2005, http://tools.ietf.org/html/draft-nottingham-http-poe-00.

[Papa07] Papazoglou, M.P.: *Web Services: Principles and Technology*, Prentice Hall, 2008.

[PaPa11] Pardon, G.; Pautasso, C.: Towards Distributed Atomic Transactions over RESTful Services, in *REST: From Research to Practice, Chapter 23*, 1. Auflage, Springer, 2011.

[Pard09] Pardon, G.: *Try-Cancel/Confirm: Transactions for (Web) Services,* 2009, http://www.atomikos.com/Publications/TryCancelConfirm.

[REST11] Doodle AG, *RESTful Doodle*. http://doodle.com/xsd1/RESTfulDoodle.pdf, Abruf am 22.04.2011.

[RiRu07] Richardson, L.; Ruby, S.: *RESTful Web Services*, O'Reilly Media, 2007.

[Somm07] Sommerville, I.: *Software Engineering*, Pearson Studium, 2007.

[Stey10] Steyer, R.: *Das JavaScript-Handbuch: Einführung, Praxis und Referenz*,

Addison-Wesley, München, 2010.

[Tane03] Tanenbaum, A.S.: *Computernetzwerke*, Pearson Studium, 2003.

[TaSt08] Tanenbaum, A.S.; van Steen, M.: *Verteilte Systeme. Grundlagen und Paradigmen*, Pearson Studium, 2008.

[Tilk09] Tilkov, S.: *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*, dpunkt Verlag, 2009.

[WKBJ03] Wittenbrink, H.; Köhler, W.; Bergmann, O.; Jung, B.; Sasaki, F.; Lenz, E.-A.; Trippel, T.; Milde, J.-T.; Poenninghaus, J.: *XML: Wissen, das sich auszahlt*, Teia Lehrbuch Verlag, 2003.

Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches entsprechend kenntlich gemacht.

Karlsruhe, 12.05.2011

Ort, Datum

Unterschrift