Eine virtuelle Maschine für den π-Kalkül zur Implementierung von Ressourcen mit dynamischen Verhalten

Olaf Märker

Hasso-Plattner-Institut für Softwaresystemtechnik
an der Universität Potsdam

- Masterarbeit -

18. Juli 2008

Abstract

The Internet affected the business world in a lasting manner. Suddenly, companies from all over the world compete against each other in the same markets. The resulting cost pressure requires a steady increase of business process automatisation. But at the same time, the complexity of these processes is growing further, making it hard to manage them manually. The π-calculus, a theory of mobile proecesses, is suitable for automatic verification and analysis of the behavior of processes in a non-static environment.

This thesis describes, how the π-calculus can be used not only for analysis, but also for the execution of process-oriented applications in the Internet. The Representational State Transfer (REST) will be used as architectural style to allow the design of highly scalable systems.

Zusammenfassung

Das Internet hat die Wirtschaft nachhaltig verändert. Unternehmen von verschiedenen Teilen der Welt konkurrieren plötzlich in denselben Märkten. Durch den dadurch erzeugten Kostendruck kommt der Automatisierung von Geschäftsprozessen eine stetig steigende Bedeutung zu. Doch gleichzeitig wächst auch die Komplexität der Prozesse, die manuell kaum noch zu bewältigen ist. Hier eignet sich der π-Kalkül, eine Theorie über mobile Prozesse, für die automatisierte Verifikation von Prozessen und der Analyse ihres Verhaltens in nicht-statischen Umwelten.

In dieser Arbeit wird beschrieben, wie der π-Kalkül nicht nur zur Analyse, sondern auch zur Ausführung von prozessorientierten Anwendungen im Internet verwendet werden kann. Als Architekturstil wird der Representational State Transfer (REST) eingesetzt, der den Aufbau eines stark-skalierbaren Systems gestattet.

Danksagung

Diese Arbeit wäre nicht ohne Hagen Overdick entstanden, der die Idee dazu hatte und viel dazu beitrug. Ich danke ihm dafür, für die vielen Diskussionen und für seine ständig neuen Ideen und Gedankengänge, denen zu folgen nicht immer einfach war. Ich danke auch Prof. Weske, der während meines Studiums die Spezialisierung in diesem Fachgebiet ermöglichte. Ich danke auch Anja Bog und Frank Puhlmann, die mit ihren Arbeiten die Grundlage für dieses Werk geschaffen haben. Frank danke ich außerdem für seine wertvolle Hilfe bei der Theorie über den π-Kalkül.

Ein ganz besonderer Dank geht auch an meine Familie, an meine Eltern für die Ermöglichung meines Studiums und für das Korrigieren, und an meine Frau und meine Kinder für ihre Geduld, ihre Unterstützung und ihre Liebe.

Potsdam, den 18. Juli 2008 Olaf Märker

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Angenommen, es existiert ein Unternehmen *XYZ* mit dem Geschäftsführer Klaus. Die Firma vertreibt verschiedenste Waren über das Internet und arbeitet dazu mit einer ganzen Reihe von Lieferanten und Herstellern zusammen. Eines Nachts hat Klaus spontan eine Idee, wie er die Abläufe im Unternehmen noch weiter optimieren könnte. Am nächsten Morgen um 9:00 Uhr im Büro angekommen, setzt sich Klaus an seinen Rechner, öffnet ein Programm, das ihm alle Prozesse im Unternehmen graphisch darstellt, er schiebt einige Aktivitäten hin und her, zieht einige neue Verbindungen zwischen ihnen, verändert noch einige Eigenschaften und drückt auf den Knopf mit der Aufschrift ‚Speichern und Ausführen'. Eine Warnung erscheint auf dem Bildschirm, in der das System ihm mitteilt, dass es nun Schwierigkeiten bei der Kommunikation mit Lieferant A geben wird. Klaus ändert also den Prozess noch einmal kurz ab und drückt erneut auf den Knopf. Die Semantik der Prozesse ist nun in Ordnung und er holt sich erst einmal einen Kaffee. Es ist 9:10 Uhr und die Kunden von XYZ erhalten ihre Waren ab heute im Durchschnitt einen Tag schneller als bisher.

Leider sind IT-Infrastrukturen meist nicht so flexibel wie bei der fiktiven Firma XYZ[1]. Doch wird in der Forschung versucht, die Automatisierung von Geschäftsprozessen immer weiter voranzutreiben. Die Ausführung von Prozessbeschreibungen durch Maschinen (zum Beispiel BPEL-Engines) ist nichts Außergewöhnliches mehr und die modellgetriebene Entwicklung und Erweiterung von

[1] Alle eventuellen Ähnlichkeiten zu real existierenden Unternehmen sind rein zufällig

prozessorientierten Anwendungen ist Gegenstand der aktuellen Forschung. [60]

Neben der Modellierung und Ausführung von Geschäftsprozessen beschäftigen sich Wissenschaftler mit der formalen Analyse der strukturellen Eigenschaften von Prozessen [52]. Über Simulationen kann etwa die Kompatibilität zweier Prozesse geprüft und mögliche Verklemmungen in der Interaktion mehrerer Prozesse vor deren Ausführung erkannt werden. Für die Beschreibung von Prozessen in einer agilen, sich ständig verändernden Umgebung, wie sie heutige Unternehmen erleben, ist der π-Kalkül [35] als Prozessalgebra besonders gut geeignet [41]. Mit ihm lassen sich dynamische Strukturen abbilden und formal analysieren [48].

Das funktioniert auch für Prozesse, die über das Internet miteinander interagieren. Das Internet ist aus der heutigen Welt nicht mehr wegzudenken und erlangte in den letzten Jahren auch in der Wirtschaft eine herausragende Bedeutung. Kommunikation mit den Kunden und Geschäftspartnern ist durch das Internet günstig und schnell geworden und ermöglicht über die Vernetzung von Softwaresystemen eine starke Automatisierung von bilateralen Abläufen. Doch gerade diese gewaltige Vernetzung birgt neue Probleme für Softwaresysteme: In einem globalen Netzwerk können Tausende von Kunden gleichzeitig auf ein System zugreifen und so viel Systemressourcen beanspruchen. Der Bekanntheitsgrad eines Dienstes kann sich schlagartig erhöhen und die Nutzerzahlen in einem kurzen Zeitraum vervielfachen. Kann die IT-Infrastruktur eines Unternehmen in einer solchen Situation nicht Schritt halten, hat dies negative Auswirkungen auf den Unternehmenserfolg.

Der Representational State Transfer (REST) [20] bietet als Architekturstil einen vielversprechenden Ansatz, diese Probleme zu überwinden und skalierbare, verteilte Anwendungen zu ermöglichen. Im Mittelpunkt von REST steht die Ressource als Entität, die über das Internet angesprochen werden kann. Der Begriff Ressource umfasst dabei alles, von einfachen Daten bis hin zu komplexen Diensten, die ausgeklügelte Geschäftsprozesse ausführen.

Durch die Forschung in diesem Gebiet scheint die oben beschriebene Vision in naher Zukunft realisierbar zu sein. Die vorliegende Arbeit soll die Entwicklung in diese Richtung vorantreiben und beschäftigt sich mit der automatischen Ausführung von Prozessen. Der Fokus dabei liegt auf Anwendungen im Internet und REST scheint als Architekturstil dafür gut geeignet zu sein. Angewandt auf die Wirtschaftsdomäne bilden Geschäftsprozesse das Verhalten von Ressourcen. Wird

eine Ressource angesprochen, so wird ein Prozess angestoßen, der die Geschäftslogik enthält und das gewünschte Verhalten erzeugt.

Dass sich der π-Kalkül für die Beschreibung und Analyse von Geschäftsprozessen eignet, hat Puhlmann bereits gezeigt [48]. Dazu werden Prozesse in den π-Kalkül überführt und durch Werkzeuge wie der Mobility Workbench [57] automatisch verifiziert. Wenn man die Prozesse bereits in den π-Kalkül überführt hat, stellt sich nun die Frage, ob diese Beschreibungen nicht auch für die Ausführung der Prozesse genügen. Wird gleichzeitig versucht, die Vorteile einer REST-Architektur auch für das Geschäftsprozessmanagement zu nutzen, ergibt sich daraus die Kernfrage dieser Arbeit:

Ist der π-Kalkül für die Implementierung von Ressourcen geeignet?

Um eine Grundlage für diese Arbeit zu schaffen, werden dazu im nächsten Kapitel die verwendeten Technologien eingeführt. Die service-orientierte Architektur (SOA) [32], der derzeit dominierende Lösungsansatz zur Vernetzung von Geschäftsprozessen, wird erklärt und darauf aufbauend die Ressourcenorientierung als neue Technologie vorgestellt, die bei der Entwicklung großer skalierbarer verteilter Anwendungen weitere Vorteile bringt. Außerdem wird der π-Kalkül eingeführt, der für die Beschreibung und Validierung von Prozessen genutzt werden soll.

In Kapitel 3 wird ein fiktiver Onlineshop als Beispiel für eine ressourcenorientierte Anwendung beschrieben, um die Ideen und Eigenschaften dieses Architekturstils zu verdeutlichen. In diesem Zusammenhang wird der Begriff Ressource genauer betrachtet. Es wird erläutert, wie Ressourcen implementiert werden können und welche entscheidende Rolle Prozesse dabei spielen. Da zur Implementierung dieser Prozesse der π-Kalkül verwendet werden soll, werden in Kapitel 4 die Grundlagen geschaffen, um den π-Kalkül internetfähig zu machen.

In Kapitel 5 wird eine virtuelle Maschine entwickelt, die die in dieser Arbeit entwickelten Konzepte beinhaltet und somit evaluiert. Gelingt die Umsetzung und entstehen dabei funktionierende Ressourcen, deren Verhalten mit dem π-Kalkül beschrieben wurde, so liefert dies eine Antwort auf die Fragestellung dieser Arbeit. Zuletzt werden ähnliche oder verwandte Arbeiten in Kapitel 6 genannt und die Ergebnisse der Arbeit im Fazit zusammengefasst.

Kapitel 2

Technologien

In diesem Abschnitt werden zunächst die dieser Arbeit zu Grunde liegenden Technologien vorgestellt. Zuerst wird die Serviceorientierung eingeführt, die derzeit im Geschäftsprozess-Management weit verbreitet ist. Danach wird die aus der Entwicklung des Internets hervorgegangene Ressourcenorientierung erläutert und der Zusammenhang zur Serviceorientierung beleuchtet. Abschließend wird mit dem π-Kalkül noch eine Prozessalgebra als Grundlage für die zu implementierende virtuelle Maschine beschrieben. Begonnen wird aber erst einmal mit einer kurzen Einführung in das Geschäftsprozessmanagement.

2.1 Business Process Management

Business Process Management (BPM, deutsch: Geschäftsprozessmanagement) umfasst das Erstellen, Analysieren, Ausführen, Überwachen und Optimieren von betrieblichen Abläufen in einem Unternehmen. Die Idee hinter BPM beschreibt Weske wie folgt:

> „Business process management is based on the observation that each product that a company provides to the market is the outcome of a number of activities performed. Business processes are the key instrument to organizing these activities and to improving the understanding of their interrelationships.“ [61]

Die Ziele des Geschäftsprozessmanagement sind vorwiegend die Profitabilität von Unternehmen durch Optimierungen der Unternehmensabläufe und deren Flexibilität zu steigern. Gerade im Zeitalter der Globalisierung und des Internets ist eine schnelle Reaktion auf Änderungen der Märkte entscheidend. Neue Produkte oder Dienstleistungen von Wettbewerbern drängen mit hoher Geschwindigkeit in Marktsegmente und konkurrieren mit den eigenen. Um zu bestehen, müssen neue Produkte zügig zur Marktreife gebracht und bestehende angepasst werden können, was eine hohe Flexibilität der Abläufe im Unternehmen erfordert.

Dafür wird jedoch ein hohes Maß an Automatisierung benötigt, weswegen das Geschäftsprozessmanagement nicht nur eine Fachrichtung der Betriebswirtschaftslehre geblieben ist, sondern auch zu einem großen Teilgebiet der Informatik avancierte. In beiden Fachbereichen werden die Prozesse untersucht, die im Unternehmen existieren. Die Begriffe Geschäftsprozess und Geschäftsprozessmanagement können nach Weske wie folgt definiert werden:

Definition 1 Ein *Geschäftsprozess* besteht aus einer Menge von Aktivitäten, die in einer organisatorischen oder technischen Umgebung koordiniert ausgeführt werden. Diese Aktivitäten realisieren zusammen ein Geschäftsziel. Jeder Geschäftsprozess wird von genau einer Organisation ausgeführt, kann aber mit Geschäftsprozessen anderer Organisationen interagieren [61]. △

Definition 2 Das *Geschäftsprozessmanagement* umfasst Konzepte, Methoden und Techniken, um Design, Administration, Konfiguration, Ausführung und Analyse von Geschäftsprozessen zu unterstützen [61]. △

Geschäftsprozesse können im Unternehmen in den unterschiedlichsten Formen definiert sein. Sie können unter anderem auf mündliche Absprachen beruhen, in textueller Form als Richtlinien oder Arbeitsanweisungen vorliegen oder auch durch Softwaresysteme umgesetzt sein. Es ist auch nicht festgelegt, wie einzelne Aktivitäten aussehen. Es können Aufgaben wie das Schreiben eines Briefes durch einen Mitarbeiter oder auch technische Tätigkeiten wie die automatisierte oder manuelle Steuerung einer Maschine oder den Aufruf eines Dienstes im Internet sein.

Von den konkreten Arten der Aktivitäten wird im Geschäftsprozessmanagement abstrahiert. Entscheidend ist die Abfolge der einzelnen Aktivitäten. Um diese

genauer zu untersuchen, werden Geschäftsprozessmodelle verwendet.

Definition 3 Ein *Geschäftsprozessmodell* ist die Abstraktion eines Geschäftsprozesses, die eine Menge von Aktivitäten und die Ausführungsabhängigkeiten zwischen ihnen darstellt. △

Ein wichtiger Punkt im Geschäftsprozessmanagement ist die Tatsache, dass meistens nicht die Möglichkeit existiert, die Prozesse in einem Unternehmen völlig neu zu entwerfen. So ist es schwierig, komplexe Änderungen bei den Mitarbeitern durchzusetzen und es müssen oftmals alte Systeme mit integriert werden, deren Neuanschaffung oder Neuentwicklung zu teuer wären.

2.1.1 Orchestrierung und Choreographie

Wie aus Definition 1 hervorgeht, wird ein Geschäftsprozess zentral von einer Organisation oder organisatorischen Einheit ausgeführt. Sie legt fest, welche Aktivitäten wann ausgeführt werden. In Anlehnung an einen Dirigenten, der ein Orchester leitet und bestimmt, wann welcher/welche MusikerIn seinen/ihren Einsatz hat, wird die Komposition von Aktivitäten auch als *Orchestrierung* bezeichnet.

Im Gegensatz dazu fehlt bei interagierenden Geschäftsprozessen verschiedener Organisationen eine zentrale Steuerungsinstanz. Jede Organisation handelt aus eigenem Antrieb, dennoch sind gewisse Regeln für die Interaktion notwendig, damit ein gemeinsam angestrebtes Ziel auch erreicht werden kann. Dieses wird als *Choreographie* bezeichnet. Wie beim Tanz, wo jeder Tänzer und jede Tänzerin wissen muss, wie sein/ihr Bewegungsablauf auszusehen hat, entsteht erst durch das Zusammenwirken ein Gesamtwerk.

Wenn es um die Automatisierung von Geschäftsprozessen geht, spielt gerade im Bereich der Choreographien das Internet eine immer entscheidendere Rolle. Es ermöglicht eine kostengünstige und vor allem schnelle Kommunikation zwischen verschiedenen Unternehmen (Business to Business (B2B)) oder auch zwischen Kunden und Unternehmen (Business to Customer (B2C)). Die vorliegende Arbeit ist in diesem Bereich des Geschäftsprozessmanagement anzusiedeln.

Im nächsten Abschnitt wird die Serviceorientierung als das derzeit vorherrschende Konzept der Interaktion zwischen Unternehmen im Internet vorgestellt.

2.2 Serviceorientierung

Die Serviceorientierung ist in der Informationstechnologie ein Paradigma zur Erstellung von verteilten Systemen, das sich an typische Dienstleistungen in der realen Welt orientiert. So gibt es Menschen oder Unternehmen, die bestimmte Bedürfnisse haben und es gibt welche, die diese Bedürfnisse befriedigen können. Hat zum Beispiel eine Person ein defektes Auto und möchte es repariert haben, so geht sie zu einer Werkstatt, die die Reparatur des Autos als Dienstleistung anbietet. Als Gegenleistung wird die Person dafür bezahlen. Das Kernkonzept bei service-orientierten Architekturen (SOA) ist der Dienst, der Angebot und Nachfrage zusammen bringt.

> „While both needs and capabilities exist independently of SOA, in SOA, services are the mechanism by which needs and capabilities are brought together.“ [32]

Die Serviceorientierung unterscheidet zwischen Dienstnutzern (Konsumenten) und Dienstanbietern. Der Anbieter erläutert seine Dienstleistung in einer *Dienstbeschreibung*. Diese enthält vor allem die Information, was für die Ausführung vom Konsumenten benötigt wird, was der Service mit diesem *Input* macht und wie das Ergebnis aussehen wird (*Output*). Ein wesentlicher Aspekt der Serviceorientierung ist die *Sichtbarkeit* der Dienste. Ein Konsument kann nur dann einen Dienst nutzen, wenn er weiß, dass er existiert und weiß, wie er ihn erreichen kann. Dafür gibt es wie auch in der realen Welt verschiedene Ansatzpunkte. Im Werkstattbeispiel könnte der Autobesitzer in einem Branchenverzeichnis wie den ‚Gelben Seiten‘ nach einer passenden Werkstatt suchen, er könnte durch Reklame in Zeitschriften oder Plakaten darauf aufmerksam werden oder er könnte die Werkstatt durch Empfehlungen von Freunden kennen. Gerade im Bereich der Webservices wird der Gelbe-Seiten-Ansatz verwendet, auf den weiter unten noch eingegangen wird.

In der Serviceorientierung wird versucht, die Schwächen von früheren Ansätzen für verteilte Systeme, wie das aus der Objektorientierung stammende CORBA [26], zu vermeiden. CORBA ist unter anderem an der schlechten Performanz bei einer Verteilung über Fernnetze hinweg gescheitert, die durch zu viel Overhead und einer feingranularen Interaktion verursacht worden ist [26]. In der Ser-

viceorientierung werden mehrere Funktionen zu einem Dienst gekapselt, sodass die Interaktion mit nur wenigen zu versendenen Nachrichten erfolgen kann. Wird das System in einem großen Netzwerk wie dem Internet mit relativ hohen Latenzzeiten betrieben, ergeben sich daraus Geschwindigkeitsvorteile. Desweiteren bietet die Trennung zwischen Dienstanbietern und Konsumenten klare Verantwortlichkeitsbereiche. In der Objektorientierung war der Client für die Erzeugung und Beseitigung von Objekten auf den entfernten System zuständig, was insbesondere bei einer Verteilung über Unternehmensgrenzen hinweg ein großes gegenseitiges Vertrauen voraussetzte. Ein Serviceprovider ist hingegen für den Betrieb seines Dienstes und für dessen Lebenzyklus allein verantwortlich. Er bestimmt, wie lange ein Dienst läuft und wann er gegebenenfalls neu gestartet wird. Das wiederum erzwingt eine zustandslose Kommunikation [40]: Der Konsument sendet alle für die Ausführung des Dienstes erforderlichen Informationen mit dem Dienstaufruf und der Dienst sendet die vollständige Antwort ebenfalls in genau einer Nachricht. Dies findet auch seine Entsprechung im Werkstattbeispiel. Hat der Kunde den Reparaturauftrag erteilt, kümmert sich die Werkstatt um die Instandsetzung des Autos und der Kunde erhält das reparierte Fahrzeug zurück. Dazwischen ist im Normalfall keine Kommunikation zwischen Kunde und Werksatt erforderlich. Auch hat der Kunde keinen Einfluss auf die Arbeitsabläufe innerhalb der Werkstatt. Dies obliegt allein dem Dienstanbieter.

Vorteile von SOA Die Vorteile des SOA-Paradigmas liegen vor allem in der Möglichkeit, relativ einfach große, skalierbare Systeme aufzubauen, die über Netzwerke miteinander interagieren, um ein gemeinsames Ziel zu erreichen. Eine strikte Trennung der Verantwortungsbereiche zwischen Dienstanbietern und -nutzern ermöglichen es, das Netzwerk über Unternehmensgrenzen hinweg zu spannen.

Serviceorientierte Architekturen können in unterschiedlichen Szenarien eingesetzt werden. Ein Haupteinsatzgebiet ist die Integration von Softwaresystemen in die IT-Infrastruktur von Unternehmen, auch Enterprise Application Integration (EAI) genannt. Durch die in der Dienstbeschreibung festgelegten Schnittstelle können neue oder übernommene Anwendungen mit relativ geringem Anpassungsaufwand die Dienste der anderen Softwaresysteme nutzen. Der Einsatz von SOA im Bereich der EAI wird in dieser Arbeit nicht weiter betrachtet werden. Weiter-

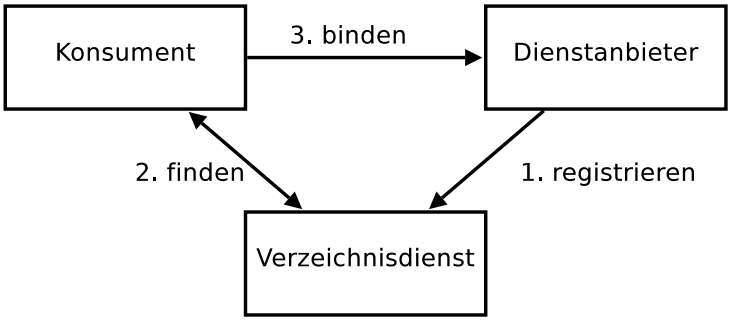

Abbildung 2.1: Aufbau einer service-orientierten Architektur

führende Informationen zu diesem Thema sind in [22, 3] zu finden.

Das zweite große Einsatzgebiet von SOA ist die Bereitstellung von Diensten im Internet, den so genannten *Webservices*, auf die im folgenden Abschnitt gesondert eingegangen wird.

2.2.1 Webservices

Webservices sind eine konkrete Ausprägung einer service-orientierten Architektur. Sie erlauben es, Dienste im Internet anzubieten und durch die Nutzung von anerkannten und verbreiteten Protokollen aufzurufen. Bei Webservices handelt es sich aber nicht um einen eigenen Standard, es ist vielmehr ein Konzept, das auf der Nutzung von mehreren Standards beruht. Definiert werden können Webservices wie folgt:

Definition 4 *Webservices* sind abgeschlossene, modulare Anwendungen, die über das Internet beschrieben, veröffentlicht, lokalisiert und aufgerufen werden können. [27] △

Um Webservices im Internet veröffentlichen zu können, gesellt sich zu den beiden Rollen Konsument und Dienstanbieter aus der Serviceorientierung noch ein Verzeichnisdienst oder Service Broker, welche die Funktion der ‚Gelben Seiten' übernehmen. Die Interaktion zwischen diesen drei Rollen ist in Abbildung 2.1 dargestellt. Die Dienstanbieter registrieren ihre Dienste bei einem Verzeichnisdienst. Benötigt nun der Konsument eine Funktionalität, die er selber nicht bereitstellen kann oder möchte, so kann er eine Anfrage an den Verzeichnisdienst stellen, um

einen Dienst zu finden, der diese Funktionalität bereitstellt. Findet sich ein passender Dienst, sendet der Verzeichnisdienst die Dienstbeschreibung inklusive der Adresse des Dienstes an den Konsumenten zurück, der sich jetzt mit dem Dienst *binden* kann.

Dabei werden typischer Weise die drei folgenden Protokolle eingesetzt:

- *SOAP* [59] ist eine leichtgewichtiges XML-Protokoll zur Übertragung von strukturierten Daten in einer dezentralisierten, verteilten Umgebung. Es wird zur Kommunikation mit dem Verzeichnisdienst und zum Nachrichtenaustausch zwischen Konsument und Dienstanbieter genutzt.

- Die Web Services Description Language (WSDL) [12] definiert ein Format, um Webservices zu beschreiben. In einem WSDL-Dokument werden die aufrufbaren Funktionen, die erforderlichen Daten und ihre Datentypen sowie die zu verwendenden Kommunikationsprotokolle aufgeführt. Ferner müssen in einer dezentralen Architektur die Kommunikationsendpunkte beschrieben werden, damit Konsumenten wissen, wo ein Service aufzurufen ist.

- Für die Kommunikation mit dem Verzeichnisdienst wurde Universal Description, Discovery and Integration (UDDI) [8] vorgesehen, welches eine standardisierte Schnittstelle zur Registrierung und Abfrage von Diensten bereitstellt.

- Zur Übertragung von SOAP-Nachrichten wird häufig *HTTP* [21] verwendet, was insbesondere den Umgang mit Firewalls erleichtert.

Webservices erlauben den Aufbau von Systemen, die nur lose miteinander gekoppelt sind. Speziell trifft dies auf die Services zu: Der Konsument ist nicht fest an den Dienst eines Anbieters gebunden sondern kann auch die Dienste anderer Anbieter nutzen, die die gleiche oder eine ähnliche Leistung anbieten. Das gestattet auch eine *späte Bindung* zwischen Konsument und Service. Welchen Service ein Konsument nutzt, muss erst zur Laufzeit entschieden werden. Ist zu diesem Zeitpunkt ein Dienst nicht verfügbar, kann der Konsument vom Verzeichnisdienst einen alternativen Dienst erhalten. Dadurch wird die Ausfallsicherheit des gesamten Systems erheblich gesteigert.

2.2.2 Webservice-Kompositionen

Die Komposition von Webservices ist ein Konzept zur Realisierung von Anwendungen durch die Nutzung von existierenden Diensten. Zum Beispiel ist es möglich, Webservices zur Buchung von Hotels und Webservices zur Buchung von Flügen zu einer Reisebuchungsanwendung zusammenzufügen. Webservice-Kompositionen besitzen insbesondere folgende Eigenschaften: Sie sind rekursiv, da eine Komposition wieder ein Webservice darstellen und in weiteren Kompositionen verwendet werden kann. Die Reisebuchungsanwendung könnte selber eine Webserviceschnittstelle besitzen, die zum Beispiel Reisebüros für ihre eigenen Kompositionen nutzen könnten. Eine Komposition ist ferner transparent gegenüber den Dienstanbietern, da sich die Dienstaufrufe innerhalb und außerhalb einer Komposition nicht unterscheiden. Eine Komposition beschreibt nur die Zusammenhänge zwischen mehreren Dienstaufrufen, die als Aktivitäten in einem Prozess aufgefasst werden können.

2.2.3 Business Process Execution Language (BPEL)

In der Industrie hat sich zur Beschreibung von Webservice-Kompositionen als Standard die Business Process Execution Language (BPEL) [30] durchgesetzt. BPEL ist eine XML-Beschreibungssprache und erlaubt es, Prozesse zu definieren, deren atomare Aktivitäten durch Webservice-Aufrufe realisiert werden. Dabei können abstrakte und konkrete Webservice-Kompositionen beschrieben werden. Abstrakte Definitionen stellen das von außen sichtbare Verhalten der Komposition dar, während konkrete Prozesse den Ablauf der Komposition komplett festlegen und sich durch einen Interpreter automatisiert ausführen lassen. Dazu existieren so genannte *BPEL-Engines*, Ausführungsmaschinen, die konkrete BPEL-Prozesse interpretieren und die im Prozess beschriebenen Webservices aufrufen, um so ein Geschäftsziel zu erreichen. Durch eine solche Automatisierung von Prozessen erhofft man sich Einsparungen im Unternehmensablauf zu erreichen.

Aufbau von BPEL

Zur Kommunikation werden in BPEL die Elemente *Invoke* zum Aufruf eines Webservice, *Receive* zum Empfangen einer Nachricht und *Reply* zum Senden einer Antwort für eine empfangene Nachricht verwendet. Ein Reply bezieht sich dabei stets auf ein Receive und so kann eine verbindungsorientierte Zweiwege-Kommunikation modelliert werden. Als weitere Aktionen bietet BPEL folgende Elemente an:

- *Assign* setzt den Wert einer Variable
- *Throw* wirft eine Fehlermeldung
- *Wait* verzögert die Ausführung
- *Exit* beendet einen Prozess

Der Kontrollfluss zwischen diesen Aktivitäten kann durch folgende Konstrukte beschrieben werden:

- *Sequence* führt Aktivitäten in der angegebenen Reihenfolge aus
- *If* führt eine Aktivität aus, wenn eine bestimmte Bedingung erfüllt ist.
- *While* führt eine Aktivität wiederholt aus, so lange eine bestimmte Bedingung erfüllt ist.
- *RepeatUntil* führt eine Aktivität wiederholt aus, bis eine bestimmte Bedingung erfüllt ist.
- *Pick* wartet auf ein Ereignis wie den Erhalt einer Nachricht oder eines Timeouts. Tritt eines der Ereignisse ein, wird eine entsprechende Aktivität ausgeführt.
- *Flow* startet mehrere Aktivitäten parallel.
- *ForEach* führt eine Aktivität mehrmals parallel aus. Die Anzahl der Ausführungen wird zuvor festgelegt.
- *Link* spezifiziert Abhängigkeiten zwischen verschiedenen Aktivitäten.

Über die aufgezählten Kontrollflusselemente lassen sich verschachtelte Blockstrukturen ausdrücken. Eine Sequence kann zum Beispiel mehrere Flow-Anweisungen oder Schleifenkonstrukte enthalten, die wiederum weitere strukturierte Elemente enthalten können. Mit den Links lassen sich darüber hinaus Abhängigkeiten zwischen den Aktivitäten bzw. Kontrollflussstrukturen angeben. Mit Links lassen sich so Graphstrukturen beschreiben, die auch über einzelne Blöcke hinaus gehen können.

Fehlerbehandlung

Außerdem bietet BPEL an, Fault- und CompensationHandler zu definieren. *CompensationHandler* ermöglichen es, die Änderungen von schon vollständig ausgeführten Prozessteilen rückgängig zu machen. So wird die Möglichkeit geboten, Prozesse zurück zu rollen, und das auch dann, wenn keine Transaktionen realisierbar sind. Ein CompensationHandler könnte zum Beispiel eine getätigte Hotelbuchung stornieren, wenn in einer Komposition die Buchung eines passenden Fluges fehl schlägt. Treten Fehler in Prozessteilen auf, die noch nicht abgeschlossen sind, können die Fehler durch *FaultHandler* behandelt werden. FaultHandler sollen nicht abgeschlossene Tätigkeiten aus den zugehörigen Bereichen rückgängig machen. Dabei können sie auch andere CompensationHandler ausführen.

Jeder Bereich eines BPEL-Prozesses kann einen CompensationHandler und mehrere FaultHandler besitzen. Durch die mögliche Schachtelung von Bereichen durch die Blockstrukturen können auch die Handler geschachtelt sein. Ein Bereich kann also seine FaultHandler besitzen und jeder Unterbereich kann trotzdem seine eigenen Handler mitbringen. Über eine *Rethrow*-Aktivität ist es möglich, dass ein FaultHandler die Behandlung einer Fehlermeldung an einen FaultHandler in einem übergeordneten Bereich abgibt.

Korrelationen

Bei der Interaktion mit komplexen Diensten sind oft mehr als zwei Nachrichten zwischen Konsument und Anbieter erforderlich. Für jeden Konsumenten wird deshalb ein eigenes Exemplar des Auszuführenden Prozesses erzeugt. Da alle Konsumenten ihre Anfragen an den in der Dienstbeschreibung genannten Kommuni-

kationsendpunkt senden, muss auf der Serverseite eine Zuordnung zwischen den Nachrichten der Konsumenten und den jeweiligen Exemplaren des Prozesse stattfinden. Diese Zuordnung wird als Korrelation bezeichnet [30]. Für die Zuordnung werden bei BPEL Identifikatoren in den Metadaten einer Nachricht eingefügt und in der Kompositionsbeschreibung wird mit *Correlation Sets* definiert, wie diese zu interpretieren sind. So könnte für eine Zuordnung von Nachrichten zu einem Bestellprozess die Kundennummer zusammen mit einer Auftragsnummer verwendet werden. Nachrichten mit derselben Kunden- und Auftragsnummer würden dann zum selben Prozessexemplar geleitet.

Die Umsetzung von verteilten Anwendungen im Internet als Webservices und deren Komposition mit BPEL ist der derzeitig verwendete Standard in vielen produktiven Systemen. Doch gerade in Bezug auf die Flexibilität von Softwarearchitekturen und deren Skalierbarkeit versuchen neuere Ansätze die Möglichkeiten von SOA zu erweitern. Als ein vielversprechender Ansatz wird im folgenden Abschnitt die Ressourcenorientierung beschrieben.

2.3 Ressourcenorientierung

In der jüngsten Zeit scheint sich ein regelrechter Hype um den Representational State Transfer (REST) [20] zu entwickeln und es existiert eine kontroverse Diskussion zwischen Webservice- und REST-Befürworter [24]. Während die einen Argumentieren, dass SOAP und die große Anzahl der darauf aufbauenden Protokolle (Oft auch als WS-* bezeichnet) zu komplex sind und sich mit REST Anwendungen nun endlich einfacher realisieren lassen, beteuern die anderen, dass erst durch die fortgeschrittene Standardisierung der WS-*-Protokolle ein Mehrwert für die Anwendungsentwicklung entsteht. In diesem Abschnitt soll die auf REST basierende Ressourcenorientierung aus der architektonischen Sicht eingeführt und die Unterschiede zu SOA dargestellt werden.

Der Representational State Transfer hat seinen Ursprung in der Weiterentwicklung des Hypertext Transfer Protocol (HTTP) [21] von der Version 1.0 zu 1.1. Bei der Entwicklung stand die Frage im Vordergrund, was das Web so erfolgreich gemacht hat, welche Probleme dennoch existieren und wie man sie beheben kann [20]. Der dabei geschaffene Architekturstil, der durch REST beschrieben wird, eignet

sich zur Realisierung von skalierbaren, verteilten Anwendungen mit globalen Ausmaßen. Kernbestandteil von REST sind die *Ressourcen*. Jedem Konzept, welchem ein Name gegeben werden kann, lässt sich als Ressource darstellen. Das kann ein Textdokument, ein Urlaubsfoto oder auch das aktuelle Wetter von Berlin sein. Der Begriff *Ressource* wurde mit Absicht so allgemein gewählt, um möglichst viele Dinge, seien sie abstrakt oder real-existierend, zu erfassen. Als Bezeichner für Ressourcen werden Uniform Resource Identifiers (URIs) [10] verwendet. Durch den identifizierenden Charakter sind URIs eindeutig, d.h. ein URI bezeichnet genau ein Konzept. Die Uniformität erleichtert den Umgang mit verschiedenen URIs in komplexen Systemen.

Ferner besitzen Ressourcen eine einheitliche Schnittstelle auf Protokollebene mit festgelegter Semantik, über die Clients mit der Ressource interagieren können. Der Datenaustausch erfolgt über *Repräsentationen*, die den aktuellen Zustand einer Ressource darstellen. Clients können über die festgelegte Schnittstelle die Repräsentation einer Ressource erfragen oder über das Senden einer neuen Repräsentation an die Ressource deren Zustand ändern. Die Repräsentation einer Ressource kann Verweise auf weitere Ressourcen enthalten (Zum Beispiel Links auf einer Webseite). Erhält ein Client eine Repräsentation, steht ihm nun frei, einen dieser Verweise zu folgen und die nächste Repräsentation zu erfragen. Er kann aber auch die Repräsentation von bereits besuchten Ressourcen erneut aufrufen. Der Zustand der Anwendung wird so vom Client verwaltet, er kann zwischen Ressourcen wechseln oder alte Zustände über Lesezeichen wieder aufrufbar machen. Die Ressourcen selber können nur mögliche Folgezustände anbieten.

Werden lang laufende Prozesse in der Ressourcenorientierung betrachtet, so stellt jedes Exemplar eines Prozesses selber eine Ressource dar und besitzt somit auch einen eigenen URI. Wird ein Exemplar eines Prozesses erzeugt, wird dem Client der URI dieses Exemplars mitgeteilt und er kann unmittelbar über die einheitliche Schnittstelle mit dem Exemplar interagieren. Eine komplexe Umsetzung von Korrelationen wie in der Serviceorientierung kann somit entfallen.

Fielding beschreibt einen Architekturstil als eine Sammlung von Einschränkungen, die gewählt werden, um bestimmte Eigenschaften eines Systems zu erreichen. [20] Eine Eigenschaft, die für Anwendungen im Internet von besonderer

Bedeutung ist, ist die Skalierbarkeit.

Definition 5 Die *Skalierbarkeit* eines Softwaresystems ist dessen Fähigkeit, unter Verwendung von mehr Hardwareressourcen wie Prozessorkapazität, Speicher oder Netzwerkbandbreite mehr Leistung bereitzustellen. Die Erhöhung der Leistung kann die Beschleunigung von Berechnungen, die Verkürzung von Antwortzeiten oder auch die Möglichkeit beinhalten, mehr Nutzer und/oder Daten gleichzeitig zu bedienen bzw. zu verarbeiten. △

Im Idealfall verhalten sich die aufgewandten Hardwareressourcen und die erreichte Leistung proportional zueinander; werden die Hardwareressourcen verdoppelt, sollte sich auch die Leistung des Systems verdoppeln. Doch aufgrund der Vielzahl der möglichen Hardwareressourcen und Leistungsparametern ist die Skalierbarkeit eines Systems nicht einfach zu bestimmen. Im Internet ist ein entscheidender Faktor die Anzahl der von einem Server gleichzeitig bedienbarer Nutzer, da diese Anzahl starken Schwankungen unterliegt und rasch anwachsen kann. Durch das Berücksichtigen der Skalierbarkeit beim Systementwurf und der Implementierung von Serversystemen wird versucht, später auftretende Überlastsituationen durch viele gleichzeitige Nutzer des Systems durch den Austausch oder der Erweiterung von Hardware zu begegnen. In vielen Fällen wirkt sich die Verwendung von gut skalierenden Algorithmen mit einer geringen Laufzeitkomplexität auch positiv auf die erreichbare Leistung auf der bestehenden Hardware aus.

REST begünstigt die Skalierbarkeit von Servern, da die Einschränkungen der Architektur so ausgewählt wurden, dass die Server möglichst wenig Hardwareressourcen benötigen. Die gewählten Einschränkungen der Architektur sind wie folgt:

- **Client-Server-Kommunikation** Eine Kommunikation wird ausschließlich von einem Client angestoßen, indem er eine Anfrage an einen Server schickt. Der Server antwortet auf die Anfrage des Clients.

- **Layered System** Das System kann in mehreren Schichten unterteilt werden, indem zwischen Client und Server weitere Komponenten eingefügt werden können. Dieses können zum Beispiel Proxys oder auch Caches sein. Diesen Zwischenstationen ist es erlaubt, die Nachrichten abzuändern, zu blockieren, umzuleiten oder selber zu beantworten.

- **Cacheable** An jeder Stelle zwischen Client und Server können Caches hinzugefügt werden, die sich Anfragen von Clients und die dazugehörigen Antworten der Server merken und bei erneuter Sendung einer Anfrage die gespeicherte Antwort an den Client senden. Dieses kann die Performanz der Kommunikation erheblich steigern, da Teile der Übertragung gänzlich entfallen können. Oft existiert neben einem Cache auf der Serverseite auch ein Cache auf der Clientseite (zum Beispiel der Browsercache) und eventuell betreibt der Internetprovider zusätzlich eigene Caches.

- **Stateless** Die Kommunikation erfolgt stets zustandslos. Der Server hält nur die Client-Daten vor, die zur Bearbeitung der aktuellen Anfrage benötigt werden. Dadurch wird der Server von der Verwaltung von Client-Sitzungen entlastet und hat mehr Hardwareressourcen für die eigentliche Bearbeitung der Anfragen frei.

- **Code on Demand** Der Server ist in der Lage, Berechnungen auf den Client zu verlagern, in dem er den auszuführenden Code mit der Antwort sendet. Das sind typischerweise Skripte, geschrieben in JavaScript, oder Flash-Anwendungen. So kann die Funktionalität des Clients erweitert werden und der Server muss nicht jede Berechnung selber ausführen.

- **Uniform Interface** Damit Zwischenstationen Nachrichten auf dem Weg vom Client zum Server und zurück manipulieren können, muss die Semantik der Nachricht nach außen erkennbar sein. Dazu erhalten alle Ressourcen eine einheitliche Schnittstelle mit festgelegter Semantik. Die Nachrichten werden als sicher, idempotent oder nicht-idempotent klassifiziert.

Für Nachrichten, die als sicher markiert sind, gilt, dass sie auf dem Server keine Zustandsänderung auslösen sollen. Sichere Nachrichten werden zum Abfragen von Repräsentationen von Ressourcen genutzt. Idempotente Nachrichten (Lateinisch: idem = der Gleiche, potentia = die Wirksamkeit) erzielen die gleiche Wirkung, egal wie oft sie wiederholt werden. Schickt ein Client eine idempotente Nachricht an einen Server, so kann diese einen Zustandswechsel auslösen. Sendet der Client die selbe Nachricht weitere Male, lösen diese Wiederholungen keine Änderungen auf dem Server mehr aus. Vom Prinzip her besitzt das Löschen einer Res-

source ein idempotentes Verhalten. Die erste Löschanfrage eines Clients wird die Ressource löschen, bei allen weiteren Löschanfragen ändert sich auf dem Server kein Zustand mehr, da die Ressource ja nicht mehr existiert. Zu beachten ist jedoch, dass sich die Antworten des Servers auch nicht ändern dürfen. Der Client erhält nach jeder Löschanfrage eine Antwort, die die erfolgreiche Löschung der Ressource bestätigt, obwohl sie bereits durch eine frühere Löschanfrage des selben Clients beseitigt wurde. Idempotente Nachrichten eignen sich hervorragend für die Fehlerbehandlung: Wenn der Client durch einen Fehler, wie dem Ausfall eines Rechners, die Antwort eines Servers nicht empfangen kann und so im Unklaren über den Erfolg einer gesendeten Änderungsanfrage ist, kann er diese einfach wiederholen, wenn sie ein idempotentes Verhalten zeigt. Wurde die Anfrage bereits vor dem Ausfall korrekt verarbeitet, hat das erneute Senden keine Auswirkungen mehr. Wurde aber die Bearbeitung durch den Ausfall gestört, dann kann sie nun erneut stattfinden. Sichere Nachrichten sind immer auch idempotent, da deren Wiederholungen jederzeit das Gleiche bewirken, nämlich keine Änderung irgendwelcher Ressourcen. Alle Nachrichten, die nicht idempotent und somit auch nicht sicher sind, werden in dieser Arbeit als *nicht-idempotente* Nachrichten bezeichnet. Diese Nachrichten können jederzeit Änderungen von Ressourcen und auch Seiteneffekte hervorrufen.

Diese Einteilung der Schnittstelle für Ressourcen ist nicht als Zwang für Serverimplementierungen anzusehen, sondern sie hat eher den Charakter eines Vertrages zwischen Client und Server. Es steht dem Serverbetreiber frei, die Schnittstelle so zu implementieren, dass zum Beispiel auch sichere Nachrichten Zustände ändern können, nur kann in diesem Fall der Client nicht dafür verantwortlich gemacht werden. Der Client formuliert mit der Wahl einer sicheren Schnittstelle zur Kommunikation seinen Willen, keine Änderungen zu beabsichtigen. Das ist auch der Grund dafür, das Suchmaschinen-Crawler, die Milliarden Ressourcen im Internet indizieren, diese nur über sichere Nachrichten ansprechen, um nicht ungewollt Änderungen hervorzurufen.

2.3.1 Hypertext Transfer Protocol

Da die Entwicklungen vom Representational State Transfer und dem Hypertext Transfer Protocol (HTTP)[21] eng miteinander verknüpft sind, verwundert es auch nicht, das HTTP die dominanteste Implementierung von REST ist. Es wird vornehmlich für das World Wide Web (WWW) [9] verwendet und stellt dort eindrucksvoll unter Beweis, dass ein großes System mit hunderten von Millionen Nutzern weltweit funktionieren kann.

Die einheitliche Schnittstelle von REST wird in HTTP durch die Anfragemethoden (Request Methods) realisiert. Hauptsächlich werden *GET* als sichere Methode, *PUT* und *DELETE* als idempotente Methoden und *POST* als nicht-idempotente Methode genutzt. GET dient zum Erfragen von Informationen und kann, da sie sicher ist, ohne Nebenwirkungen auszulösen auf jeder Ressource aufgerufen werden. Dadurch bietet HTTP einen universellen Reflektionsmechanismus [40]. Mit PUT kann eine neue Repräsentation an eine Ressource gesendet und mit DELETE eine Ressourcen gelöscht werden. Für alle weiteren Zwecke wird POST verwendet.

Auch in der Serviceorientierung wird HTTP zur Übertragung von SOAP-Nachrichten verwendet, jedoch geschieht das in einer nicht REST-konformen Art und Weise. Die SOAP-Spezifikation in der Version 1.1 [58] definiert formal nur die Nutzung der Methode HTTP-POST. Abhilfe schafft erst die neue Version 1.2 [59], die auch die Verwendung der anderen HTTP-Verben unter Berücksichtigung ihrer Semantik vorsieht.

2.3.2 Unterschiede zur Serviceorientierung

Wie oben dargelegt, kann ein Architekturstil durch die Wahl bestimmter Einschränkungen die Eigenschaften von Systemen bestimmen. Overdick greift diese These auf und argumentiert, dass die Ressourcenorientierung aus der Serviceorientierung durch das Hinzufügen weiterer Einschränkungen abgeleitet werden kann. Die Einschränkungen, die für SOA gelten, existieren auch für Ressourcen: sie haben eine eindeutige Referenz, einen unabhängigen Lebenzyklus und die Interaktion ist grobgranular und erfolgt zustandslos [40]. Die Ressourcenorientierung führt als weitere Einschränkung die festgelegte Schnittstelle für alle Ressourcen ein. Dies

ermöglicht die Klassifikation von Nachrichten nach ihrer Semantik, ohne den genauen Nachrichteninhalt zu kennen. Diese Zusatzinformation erlauben zum Beispiel das Cachen von Antworten für sichere oder idempotente Nachrichten. Die Einheitlichkeit der Schnittstelle aller Ressourcen verringert auch die Eintrittsbarrieren für Clients in ein System. Muss in der Serviceorientierung ein Client erst eine Dienstbeschreibung studieren, um die Kommunikationsendpunkte, die zur Verfügung stehenden Funktionen und die erforderlichen Parameter und deren Typen zu erfahren, um auch nur minimal mit einem Service zu interagieren, genügt in der Ressourceorientierung für den Anfang ein URI einer Ressource und eine einfache HTTP-GET-Nachricht, um eine Repräsentation dieser Ressource zu erhalten. Menschen sind in der Lage, mit einen normalen Browser auf Ressourcen zuzugreifen, ohne sich mit Protokollen wie WSDL oder SOAP auseinander zusetzen. Benötigt eine Ressource weitergehende Daten vom Client, kann die Repräsentation der Ressource darauf hinweisen und zum Beispiel ein HTML-Formular für die Datenerfassung bereitstellen.

Der Ansatz der ressourcenorientierten Architektur (ROA) ist in einigen Punkten pragmatischer als SOA und von den verwendeten Protokollen bei weitem nicht so komplex und sorgt so vor allem bei kleineren Projekten für Begeisterung aber auch große Unternehmen wie Google [23] und Amazon [4] setzen bereits REST ein. Auch das Business Process Management wird sich in Zukunft sehr wahrscheinlich der Ressourcenorientierung aufschließen. Einige Hersteller werben bei ihren BPM-Tools bereits mit der Unterstützung von REST oder haben dieses in Zukunft vor [14, 7]. Auch fließen die Erkenntnisse von REST in die Überarbeitung von WS-*-Protokollen ein. So gestattet SOAP in der Version 1.2 die Verwendung von HTTP-GET zum Erfragen von SOAP-Antworten und WSDL in der Version 2.0 erlaubt die Zuordnung von Verben wie GET und POST zu einzelnen Operationen und die Deklaration von sicheren Operationen. Außerdem lassen sich dynamisch erzeugte Dienste beschreiben [19]. So wie es im Moment aussieht, dürfte der Representational State Transfer das Business Process Management in Zukunft noch nachhaltig beeinflussen.

2.4 Der π-Kalkül

Der π-Kalkül [35] ist eine Prozess-Algebra und liefert eine theoretische Grundlage für Systeme von konkurrierenden Prozessen, die untereinander kommunizieren. Im Gegensatz zu anderen etablierten Prozesstheorien, wie zum Beispiel Petri-Netze, ermöglicht der π-Kalkül sich verändernde Strukturen zu modellieren. Dazu nutzt der π-Kalkül *Link Passing Mobility*, eine Mobilität die durch die Übertragung von Referenzen zustande kommt. Mit dieser Eigenschaft ist der π-Kalkül besonders dafür geeignet, Geschäftsprozesse in einer nicht-statischen Umgebung wie dem Internet zu beschreiben. Puhlmann [48] zeigt außerdem, dass alle geläufigen Workflowpattern [2] mit dem π-Kalkül ausdrückbar sind.

Der π-Kalkül beschreibt Prozesse, die über *Namen* miteinander kommunizieren. Es existiert eine unendliche Menge von Namen $\mathcal{N}$ und eine Menge $\mathcal{K}$ von Agentenbezeichnern. a, b, c, …laufen über $\mathcal{N}$ und A, B, C, …laufen über $\mathcal{K}$. Prozesse werden mit P, Q, R, …bezeichnet. Bei der Kommunikation wird ein Name als Kanal verwendet (Subjekt) und es kann ein Tupel von Namen übertragen werden (Objekt). Tupel von Namen werden durch $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, …veranschaulicht. Die Anzahl der Namen eines Tupels $\tilde{a}$ ist $|\tilde{a}|$. Kommunikation kann immer nur zwischen zwei nebenläufigen Prozessen stattfinden, bei der ein Prozess ein Objekt in den Kanal schreibt $\overline{x}\langle\tilde{y}\rangle$ und der andere Prozess vom selben Kanal liest $x(\tilde{z})$. Dabei muss die Anzahl der Namen in den Objekten $\tilde{y}$ und $\tilde{z}$ identisch sein ($|\tilde{y}| = |\tilde{z}|$) und alle Namen in $\tilde{z}$ müssen disjunkt sein.

$$\overline{x}\langle\tilde{y}\rangle.P \mid x(\tilde{z}).Q \longrightarrow P \mid Q\{^{\tilde{y}}/_{\tilde{z}}\} \tag{2.1}$$

In Gleichung 2.1 können zwei Prozesse durch die *parallele Komposition* $\mid$ über den Kanal x miteinander kommunizieren. Bei der Leseoperation dienen die Namen in $\tilde{z}$ als Platzhalter und werden durch die empfangenen Namen ersetzt. Nach der Kommunikation bleiben P und Q übrig, wobei in $Q\{^{\tilde{y}}/_{\tilde{z}}\}$ die Namen in $\tilde{z}$ durch die Namen in $\tilde{y}$ ersetzt worden sind. Der Punkt . bezeichnet eine Sequenz, so dass $\overline{x}\langle\tilde{y}\rangle$ vor P ausgeführt werden muss. Da die Kommunikation synchron verläuft, kann eine Sendeoperation erst ausgeführt werden, wenn ein anderer Prozess vom selben Kanal versucht zu lesen und umgekehrt. Das Lesen und Schreiben blockiert also die weitere Ausführung des Prozesses, bis eine Kommunikation

stattfinden kann (*blockierende Semantik*).

Freie und gebundene Namen Namen können an einen Prozess gebunden werden, das heißt, dass sie außerhalb des Prozesses keine Relevanz besitzen. Namen werden durch den Leseoperator und den Restriktionsoperator an einen Prozess gebunden. So können in Gleichung 2.2 die ersten beiden parallelen Prozesse nicht über z kommunizieren, da das z im zweiten Prozess gebunden ist und nur als Platzhalter dient. Bei der Kommunikation zwischen den zweiten und dritten Prozess wird dieses z durch y ersetzt. Die $\mathbf{0}$ terminiert einen Prozess, sie wird oft jedoch nicht explizit aufgeführt.

$$\overline{z}\langle a\rangle.\mathbf{0} \mid x(z).z(b).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0} \longrightarrow \overline{z}\langle a\rangle.\mathbf{0} \mid y(b).\mathbf{0} \mid \mathbf{0} \tag{2.2}$$

Durch den Restriktionsoperator $\mathbf{v}$ werden neue Namen erzeugt, die von jedem anderen Namen im System verschieden sind. Der Geltungsbereich dieser Namen beschränkt sich auf den Prozess, in dem sie erzeugt wurden und können so nicht mehr zur Kommunikation mit anderen Prozessen genutzt werden. Jedoch sind interne Kommunikationen, wie in Gleichung 2.3 dargestellt, möglich, welche durch die Einschränkung der Namen nicht durch andere Prozesse behindert werden kann.

$$\mathbf{v}c\ (\overline{c}\langle\tilde{y}\rangle.\mathbf{0} \mid c(\tilde{z}).\mathbf{0}) \tag{2.3}$$

Alle Namen in einem Prozess, die nicht gebunden sind, sind *frei*. Die Funktion $bn(P)$ gibt die gebundenen Namen und die Funktion $fn(P)$ die freien Namen eines Prozesses P an. So ist $bn(x(y).\mathbf{v}z\ \overline{y}\langle z\rangle) = y, z$ und $fn(x(y).\mathbf{v}z\ \overline{y}\langle z\rangle) = x$.

Der Geltungsbereich von restringierten Namen kann durch das Senden dieser Namen an andere Prozesse erweitert werden. In Gleichung 2.4 erzeugt der erste Prozess den Namen y und sendet ihn über x an den zweiten Prozess, wo dann a durch y ersetzt wird. Danach ist der Name y beiden Prozessen bekannt und kann unter ihnen zur Kommunikation verwendet werden. Der zweite Prozess legt ebenfalls einen Namen an (b) und teilt diesen dem ersten Prozess mit. Die Restriktion der Namen y und b umschließt anschließend beide Prozesse P und Q. Nur diesen beiden Prozessen sind die Namen bekannt.

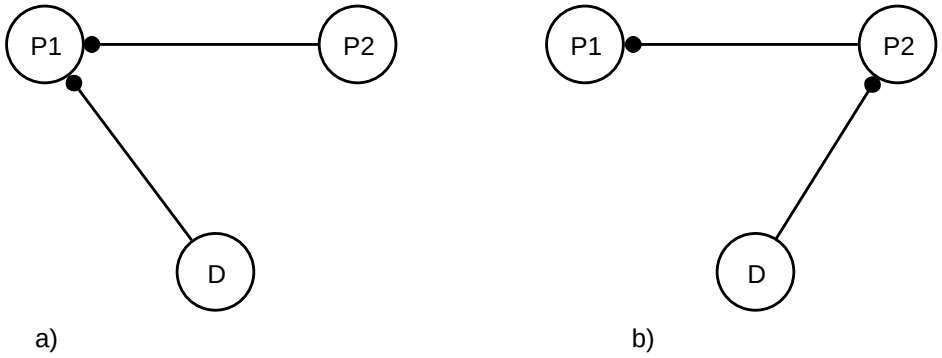

Abbildung 2.2: Link Passing Mobility

$$\begin{aligned} \mathbf{v}y\ (\overline{x}\langle y\rangle.y(z).P) \mid x(a).\mathbf{v}b\ (\overline{a}\langle b\rangle.Q) &\longrightarrow \\ \mathbf{v}y\ (y(z).P \mid \mathbf{v}b\ (\overline{y}\langle b\rangle.Q\{^{y}/_{a}\})) &\longrightarrow \\ \mathbf{v}y, b\ (P\{^{b}/_{z}\} \mid Q\{^{y}/_{a}\}) & \end{aligned} \tag{2.4}$$

Durch das Erzeugen und Versenden von Namen lassen sich viele Konzepte aus der realen Welt beschreiben. Wird ein Name als Adresse eines Prozesses angesehen (‚Über diesen Namen muss ich eine Nachricht schicken, um mit dem Prozess zu kommunizieren‘), benötigt man vor der Kommunikation den Namen des Prozesses, so wie wir erst die Telefonnummer eines Gesprächspartners benötigen, um ihn anzurufen. Die benötigten Namen können durch vorausgehende Kommunikationen in Erfahrung gebracht werden. Es lassen sich so Prozesse beschreiben, bei denen die Kommunikationspartner vor der Abarbeitung noch nicht bekannt sind. Das erlaubt zum Beispiel die Modellierung von Systemen mit einer dynamischen Auffindung von Diensten. [41] Diese Eigenschaft des π-Kalküls wird als *link-passing Mobility* bezeichnet. Ein Prozess wird mobil, da sich seine bekannte Umwelt durch die Übertragung von Namen ändern kann.

In Abbildung 2.2 a) ist ein Prozess $P1$ dargestellt, der Referenzen zu einem Prozess $P2$ und zu einem Prozess D besitzt. D könnte zum Beispiel ein Drucker sein und $P1$ kann über die Referenz auf den Drucker zugreifen. $P2$ kennt den Drucker nicht. Nun kann $P1$ die Referenz auf den Drucker an $P2$ übermitteln und $P2$ kann selber mit dem Drucker kommunizieren b). Im Sinne der Mobilität hat sich

der Drucker durch die Übertragung von Referenzen von $P1$ zu $P2$ ‚bewegt'.

Kontrollfluss Neben der parallelen Komposition und der sequentiellen Anordnung von Kommunikation existieren noch eine exklusive Auswahl (*Summation*), ein Vergleichsoperator und die Möglichkeit, definierte Agenten aufzurufen. Bei der Summation (dargestellt als $+$) wird stets einer der angegebenen Prozesse ausgeführt und alle anderen werden verworfen. Gewählt wird der Prozess, der als erstes ausgeführt werden kann. In der Summation in Gleichung 2.5 versucht ein Prozess von x und einer von y zu lesen. Beide Leseversuche blockieren so lange, bis ein geeigneter Kommunikationspartner existiert. Im Beispiel sendet ein weiterer Prozess den Namen c über y und so wird in der Summation der zweite Pfad gewählt und Q ausgeführt. Der erste Pfad wird hingegen eliminiert. Sind zwei Pfade einer Summation gleichzeitig ausführbar wie in $x(a).P + y(b).Q \mid \overline{y}\langle c\rangle \mid \overline{x}\langle d\rangle$, ist nicht festgelegt, welcher der Pfade gewinnt.

$$x(a).P + y(b).Q \mid \overline{y}\langle c\rangle \longrightarrow Q\{^{c}/_{b}\} \mid 0 \tag{2.5}$$

Der Vergleichsoperator $[x = y]$ prüft zwei Namen auf Gleichheit. Der Nachfolgende Prozess wird nur ausgeführt, wenn beide Namen etwa durch eine Ersetzung $\{^{x}/_{y}\}$ identisch sind, andernfalls wird er verworfen. In Gleichung 2.6 wird durch die Kommunikation zwischen den beiden parallelen Prozessen a durch b ersetzt und dadurch ist der erste Vergleich positiv während der andere negativ ausgeht.

$$\begin{aligned} x(a)([a = b]\overline{y}\langle d\rangle + [a = c]\overline{y}\langle e\rangle) \mid \overline{x}\langle b\rangle &\longrightarrow \\ [b = b]\overline{y}\langle d\rangle + [b = c]\overline{y}\langle e\rangle \mid \mathbf{0} &\longrightarrow \\ \overline{y}\langle d\rangle \mid \mathbf{0} & \end{aligned} \tag{2.6}$$

Agenten bestehen aus einem Bezeichner und einem Prozess. Bei der Definition von Agenten müssen alle freien Namen im Prozess als Parameter des Agenten aufgeführt werden und diese Namen müssen disjunkt sein.

$$A(\tilde{x}) \stackrel{def}{=} P \tag{2.7}$$

Tabelle 2.1: Die Grammatik des π-Kalküls

$$P ::= M \mid P|P' \mid \mathbf{v}z\ P \mid !P \mid A(\tilde{a}) \tag{2.10}$$
$$M ::= 0 \mid \pi.P \mid M + M' \tag{2.11}$$
$$\pi ::= \overline{x}\langle y\rangle \mid x(z) \mid \tau \mid [x = y]\pi \tag{2.12}$$

Aus einem Prozess heraus kann jeder Agent aufgerufen werden. Dies ermöglicht auch Rekursion, da der Prozess eines Agenten den selben Agenten wieder aufrufen kann.

$$A(x, y) \stackrel{def}{=} \overline{x}\langle y\rangle.x(z).A(x.z) \tag{2.8}$$

Ferner können Prozesse repliziert werden (dargestellt durch !). Bei der Replikation werden so viele Kopien eines Prozesses angelegt, wie benötigt werden. In Gleichung 2.9 wird der erste Prozess für die Kommunikation über x einmal benötigt und deshalb einmal kopiert. Die Kommunikation erfolgt dann mit der Kopie, das Original mit dem Replikationsoperator bleibt unverändert erhalten und kann bei Bedarf weiter repliziert werden.

$$!\overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow !\overline{x}\langle y\rangle.P \mid \overline{x}\langle y\rangle.P \mid x(z).Q \longrightarrow !\overline{x}\langle y\rangle.P \mid P \mid Q\{^{y}/_{z}\} \tag{2.9}$$

Der π-Kalkül betrachtet nur die Kommunikation zwischen Prozessen, alle anderen Aspekte der Prozessausführung werden durch τ abstrahiert. Das τ kann zum Beispiel interne Kommunikation darstellen und wird später in dieser Arbeit zur Einbindung von funktionalen Aspekten verwendet. Ein τ kann immer ausgeführt werden, es besitzt also kein blockierendes Verhalten. So wird aus $\tau.P \longrightarrow P$.

Aus allen beschriebenen Elementen setzt sich die Grammatik des π-Kalküls wie in Tabelle 2.1 gezeigt zusammen.

2.4.1 Strukturelle Kongruenz

Von entscheidender Bedeutung für die Theorie ist die Frage, wann zwei Prozesse als gleich anzusehen sind. Für kommunizierende Prozesse ist das Verhalten wichtig, das dieser Prozess zeigt. Das Verhalten ist jedoch unabhängig von der Wahl der gebundenen Namen. So kann man die Prozesse $P = x(a).a(y)$ und $Q = x(b).b(y)$ nicht vom Verhalten her unterscheiden, da die Struktur beider Prozesse gleich ist und sie sich nur in den gebunden Namen unterscheiden. P und Q sind *strukturell kongruent*, $P \equiv Q$.

Für die Entscheidung auf strukturelle Kongruenz gelten folgende Regeln [34]:

1. Prozesse sind identisch, wenn sie sich nur in den gebundenen Namen unterscheiden.
2. $P + 0 \equiv P$
3. $P \mid 0 \equiv P$
4. $!P \equiv P \mid !P$
5. $\mathbf{v}x\ 0 \equiv 0$
6. $\mathbf{v}x\ \mathbf{v}y\ P \equiv \mathbf{v}y\ \mathbf{v}x\ P$
7. Wenn $x \notin bn(P)$, dann $\mathbf{v}x\ (P \mid Q) \equiv P \mid (\mathbf{v}x\ Q)$
8. $[x = x]\pi.P \equiv \pi.P$
9. $A(\tilde{y}) \equiv P\{^{\tilde{y}}/_{\tilde{x}}\}$, wenn $A(\tilde{x}) = P$

Die Summation und die parallele Komposition von Prozessen ist sowohl kommutativ als auch assoziativ, so dass $M + N \equiv N + M$ und auch $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ gilt.

2.4.2 Reduktionssemantik

Die Regeln zur Abarbeitung von π-Prozessen werden als *Reduktionssemantik* bezeichnet. Im Gegensatz zu anderen Prozesstheorien wie Petri-Netze [1], wo ein statischer Prozess durch die Änderung von Attributen wie der Markierung von Stellen

weiterentwickelt wird, entspricht im π-Kalkül jeder Zustand einem neuen Prozess. Das heißt, dass bei jedem Schritt in der Ausführung der Prozess verändert bzw. reduziert wird.

Ein Schritt bei der Reduktion eines Prozesses wird durch eine *Transition* $\longrightarrow$ dargestellt. Sofern zur Anwendung von Regeln eine Vorbedingung erfüllt sein muss, wird sie wie folgt dargestellt:

$$\frac{Vorbedingung}{Schlussfolgerung}$$

Das Hauptaxiom der Reduktionssemantik beschreibt die Kommunikation zweier paralleler Prozesse und den Wegfall anderer Alternativen von Summationen:

$$(x(\tilde{y}).P + M) \mid (\overline{x}\langle\tilde{z}\rangle.Q + N) \longrightarrow P\{^{z}/_{y}\} \mid Q \; mit |\tilde{y}| = |\tilde{z}| \tag{2.13}$$

Daneben existieren noch weitere Axiome. Die Gleichungen 2.14 und 2.15 beschreiben, dass eine Reduktion auch unter der parallelen Komposition und unter einer Restriktion möglich ist.

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \tag{2.14}$$

$$\frac{P \longrightarrow P'}{\mathbf{v}z \; P \longrightarrow \mathbf{v}z \; P'} \tag{2.15}$$

Die Regel 2.16 erlaubt die Reduktion von τ, wie sie oben beschrieben wurde.

$$\tau.P \longrightarrow P \tag{2.16}$$

Ferner lassen sich alle Prozesse reduzieren, zu denen sich strukturell kongruente Prozesse finden lasse, die nach einer der hier angegebenen Axiome reduzierbar sind (2.17).

$$\frac{Q \equiv P \quad P \longrightarrow P' \quad Q' \equiv P'}{Q \longrightarrow Q'} \tag{2.17}$$

2.4.3 Beschriftete Transitionssysteme

Die Reduktionssemantik beschreibt, wie sich Prozesse intern weiterentwickeln. Sie ist jedoch unzureichend, um das externe Kommunikationsverhalten von Agenten in einer Umwelt zu untersuchen. Für die Beobachtung von außen lässt die Reduktionssemantik nur den Schluss zu, dass etwas passiert ist, jedoch nicht, was passierte. Als Beispiel sei hier folgender Prozess gegeben:

$$\mathbf{v}y\ i(x).\tau.\overline{o}\langle y\rangle.\mathbf{0}$$

Dieser Prozess kann nach den Regeln der Reduktionssemantik nicht weiterentwickelt werden. Er könnte aber über die Namen i und o mit seiner Umwelt kommunizieren. Um externe beobachtbare Verhalten von π-Prozessen zu beschreiben, kann ein beschriftetes Transitionssystem (Labeled Transition System (LTS)) genutzt werden.

Definition 6 Ein *beschriftetes Transitionssystem* ist ein Tripel $(S, T, \xrightarrow{t})$ mit:

- einer Menge von Zuständen S
- einer Menge von Transitionsbeschriftungen T und
- $\xrightarrow{t} \subseteq S \times S$ als eine Menge von binären Transitionsrelationen für alle $t \in T$.

△

Über ein LTS lassen sich die auftretenden Aktionen bei der Kommunikation des π-Kalküls unterscheiden.

Definition 7 Die *Aktionen* α des π-Kalküls sind:

$$\alpha ::= \overline{x}\langle(\mathbf{v}\tilde{z}\ )\tilde{y}\rangle \mid x(\tilde{y}) \mid \tau$$

Akt bezeichnet die Menge der Aktionen und $\tilde{z} \subseteq \tilde{y}$. △

Die möglichen Aktionen im π-Kalkül entsprechen den Präfixen. Namen können gesendet oder empfangen werden. τ symbolisiert eine interne Aktion eines Agenten. Beim Senden können auch die restringierten Namen in $\tilde{z}$ übertragen werden und so ihr Geltungsbereich erweitert werden (Scope Extrusion). Die zugehörigen Transitionsregeln des π-Kalküls werden in Tabelle 2.2 dargestellt.

Tabelle 2.2: Die Transitionsregeln des π-Kalküls

$$STRUKT\ \frac{Q \equiv P\ P \longrightarrow P'\ P' \equiv Q'}{Q \longrightarrow Q'}$$

$$PRFIX\ \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$SUM\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$PAR\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}(bn(\alpha) \cap fn(Q) = \emptyset)$$

$$KOMM\ \frac{P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P'\ Q \xrightarrow{x(\tilde{z})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{{}^{\tilde{y}}/_{\tilde{z}}\}}(|\tilde{y}| = |\tilde{z}|)$$

$$RES\ \frac{P \xrightarrow{\alpha} P'}{\mathbf{v}z\ P \xrightarrow{\alpha} \mathbf{v}z\ P'}(z \notin n(\alpha))$$

$$OFFEN\ \frac{P \xrightarrow{\overline{x}\langle(\mathbf{v}\tilde{z}\,)\tilde{y}\rangle} P'}{\mathbf{v}\alpha\ P \xrightarrow{\overline{x}\langle(\mathbf{v}a\tilde{z}\,)\tilde{y}\rangle} P'}(x \neq \alpha \wedge \alpha \notin \tilde{z} \wedge \alpha \in \tilde{y})$$

Die Transitionsregeln für das beschriftete Transitionssystem sind in Tabelle 2.2 dargestellt. Die Regel $STRUKT$ besagt, dass wie bei der Reduktionssemantik Prozesse eine mögliche Transition besitzen, wenn eine solche Transition auch für strukturell kongruente Prozesse existiert.

Die Regel $PRFIX$ besagt, dass ein Präfix (eine Kommunikation oder ein τ) abgearbeitet werden kann. Die dabei ausgeführte Aktion α wird an die Transition geschrieben. Sie ist nach außen hin sichtbar. Dabei wird angenommen, dass diese Kommunikationen mit der Umwelt möglich sind. SUM und PAR beschreiben die Abarbeitung von Summationen und die Möglichkeit der Ausführung von Aktionen unter Parallelität.

Über die Regel $KOMM$ können zwei nebenläufige Prozesse miteinander kommunizieren. Von außen betrachtet stellt dies eine interne Kommunikation dar und es wird als Aktion nur ein τ sichtbar. Ein Beobachter kann nur feststellen, dass etwas passiert, aber es ist nicht ersichtlich, was passiert.

Die Regel RES erlaubt die Ausführung einer Aktion auch unter einer Restriktion, wenn keiner der Namen in α zu den restringierten Namen gehört. Ist dies der Fall, kommt die Regel $OFFEN$ zum Tragen. Wird ein restringierter Name versendet, dann fällt die Restriktion weg, da der restringierte Name nach der Kommunikation der Umwelt bekannt ist.

2.4.4 Bisimulationen

Die Theorie des π-Kalküls erlaubt es, das Verhalten von Prozessen miteinander zu vergleichen. Für das Geschäftsprozessmanagement sind zum Beispiel Aussagen entscheidend, wie: Prozess A verhält sich in einer Umgebung U genauso wie Prozess B und diese können deshalb gefahrlos untereinander ausgetauscht werden.

Kann der Prozess A das gesamte Verhalten von Prozess B nachahmen, dann simuliert A B. Kann zusätzlich B das Verhalten von A simulieren, dann spricht man von einer Bisimulation. Es können verschiedene Arten von Bisimulationen unterschieden werden.

Grund-Bisimulation

Die einfachste Form der Bisimulation wird als Grund-Bisimulation bezeichnet und kann wie folgt definiert werden:

Definition 8 Die *Grund-Bisimulation* ist eine symmetrische binäre Relation $\mathcal{R}$ zwischen Agenten bei der $\forall \alpha \in Akt$ gilt:

$$P\mathcal{R}Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}Q' \text{ mit } bn(\alpha) \cap (fn(P) \cup fn(Q)) = \emptyset$$

$\triangle$

Sind P und Q grund-bisimular ($P \sim Q$), dann ist für jede Aktion α das Ergebnis der beiden Prozesse wieder grund-bisimular, das heißt, es zeigt bei jedem möglichen α wieder dasselbe Verhalten. Jedoch bewertet die Grund-Bisimulation auch interne Aktionen (τ) streng, so dass das Verhalten der beiden Prozesse

$$P \stackrel{def}{=} \mathbf{v}x\ i(y).\tau.\overline{o}\langle x\rangle \tag{2.18}$$

und

$$Q \stackrel{def}{=} \mathbf{v}x\ i(y).\tau.\tau.\overline{o}\langle x\rangle \tag{2.19}$$

nicht als identisch betrachtet wird. Die Transitionsfolge von P ist $\xrightarrow{i(y)}\xrightarrow{\tau}\xrightarrow{\overline{o}\langle(\mathbf{v}x\ )x\rangle}$. Das Verhalten von Q zeigt jedoch stets zwei τ-Transitionen und kann so das Verhalten von P nicht simulieren. Werden die Prozesse P und Q als Dienste betrachtet, die auf einer Eingangsnachricht y eine Ausgangsnachricht x produzieren, dann ist dem externen Betrachter oft egal, wie viele interne Schritte zur Vollbringung der Dienstleistung notwendig sind. Dieser Umstand wird in der schwachen Grund-Bisimulation berücksichtigt.

Schwache Grund-Bisimulation

Die schwache Grund-Bisimulation abstrahiert von den internen Aktionen der Agenten. Dazu wird die Transition $\Longrightarrow$ so definiert, dass sie keine oder mehrere interne Aktionen umfasst ($\xrightarrow{x}{}^{*}$). Ferner wird $\stackrel{\alpha}{\Longrightarrow}$ als $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$ definiert und $\stackrel{\hat{\alpha}}{\Longrightarrow}$ als $\stackrel{\alpha}{\Longrightarrow}$ wenn $\alpha \neq \tau$, andernfalls als $\Longrightarrow$. Somit kann die schwache Grund-Bisimula-

tion wie folgt beschrieben werden.

Definition 9 Die *schwache Grund-Bisimulation* ist eine symmetrische Relation $\mathcal{R}$ zwischen Agenten, bei der $\forall \alpha \in Akt$ gilt:

$$P\mathcal{R}Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q' \wedge P'\mathcal{R}Q' \text{ mit } bn(\alpha) \cap (fn(P) \cup fn(Q)) = \emptyset$$

△

Die beiden Prozesse P und Q von oben besitzen nun beide die Transitionsfolge $\xrightarrow{i(y)} \stackrel{\hat{\alpha}}{\Longrightarrow} \xrightarrow{\overline{o}\langle(\mathbf{v}x\ )x\rangle}$. Der Prozesse P wird so von Prozess Q schwach grund-bisimuliert ($P \approx Q$).

Eine Einschränkung der Grund-Bisimulationen ist, dass sie keine Ersetzungen berücksichtigen. Die beiden Prozesse

$$P \stackrel{def}{=} i(x).\mathbf{0} \tag{2.20}$$

und

$$Q \stackrel{def}{=} i(x).[x = y].\overline{o}\langle x\rangle.\mathbf{0} \tag{2.21}$$

sind Grund-Bisimular ($P \sim Q$) da x und y immer als verschieden angesehen werden. Das bei der Kommunikation x durch y ersetzt werden könnte, wird von der offenen Bisimulation berücksichtigt.

Offene Bisimulation

Eine Ersetzung σ ist eine Funktion, die Namen Namen zuordnet. Eine gültige Ersetzung ist zum Beispiel $\{^{\tilde{z}}/_{\tilde{y}}\}$ wobei die Namen in $\tilde{y}$ paarweise disjunkt sein müssen und $|\tilde{y}| = |\tilde{z}|$.

Bei der offenen Bisimulation werden beim Vergleich zweier Prozesse alle möglichen Ersetzungen berücksichtigt.

Definition 10 Die *offene Bisimulation* für einen π-Kalkül ohne Restriktionen ist eine symmetrische, binäre Relation $\mathcal{R}$ zwischen Agenten bei der $\forall \alpha \in Akt$ und $\forall \sigma$ gilt:

$$P\mathcal{R}Q \wedge P\sigma \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q\sigma \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}Q'$$

△

P und Q sind offen bisimular ($P \sim_O Q$).

Die offene Bisimulation gilt nur für einen Kalkül ohne Restriktionen, da die Restriktionen problematisch sind. Werden die Prozesse

$$P \stackrel{def}{=} \mathbf{v}x\, \overline{a}\langle x\rangle.[x = y]\tau.\mathbf{0} \tag{2.22}$$

und

$$Q \stackrel{def}{=} \mathbf{v}x\, \overline{a}\langle x\rangle.\mathbf{0} \tag{2.23}$$

betrachtet, dann entwickeln sich $P \stackrel{\overline{a}\langle \mathbf{v}x\,\rangle}{\longrightarrow} [x = y]\tau.\mathbf{0}$ und $Q \stackrel{\overline{a}\langle \mathbf{v}x\,\rangle}{\longrightarrow} \mathbf{0}$. Durch die Regel $OFFEN$ wird durch das Senden des gebundenen Namens x die Restriktion aufgehoben. Dennoch ist für P immer $x \neq y$ und beide Prozesse zeigen dasselbe Verhalten. Aus Sicht der offenen Bisimulation könnte sich der Prozess P nach dem Senden von x durch eine Substitution $\{^y/_x\}$ weiterentwickeln, so dass $P \not\sim_O Q$.

Eine Möglichkeit, dieses Problem zu lösen, ist das Pflegen einer Liste von Namen, die als unterschiedlich bekannt sind.

Definition 11 Eine *Unterscheidung* $\mathcal{D}$ (Distinction) ist eine endliche, symmetrische, irreflexive, binäre Relation auf Namen. △

Sind $(a, b) \in \mathcal{D}$, so sind sie voneinander verschieden. Eine Ersetzung σ respektiert die Unterscheidung, wenn $(a, b) \in \mathcal{D} \Rightarrow \sigma(a) \neq \sigma(b)$. Mit Unterscheidungen kann die offene Bisimulation unter Berücksichtigung von Restriktionen neu definiert werden.

Offene D-Bisimulation

Definition 12 Die *offene D-Bisimulation* ist eine mit Unterscheidungen indizierte Familie aus einer Menge von symmetrischen, binären Relationen $\mathcal{R}_D$ zwischen Agenten, so dass $\forall\alpha \in Akt$ und $\forall\sigma$, die $\mathcal{D}$ respektieren, gilt:

$$P\mathcal{R}_D Q \wedge P\sigma \xrightarrow{\alpha} P' \wedge bn(\alpha) \cap (fn(P\sigma) \cup fn(Q\sigma)) = \emptyset \Rightarrow$$

1. Wenn $\alpha = \overline{x}\langle(\mathbf{v}\tilde{z}\,)\tilde{y}\rangle$ dann $\exists Q' : Q\sigma \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}_{D'}Q'$ mit $D' = D\sigma \cup \{\{z\} \times (fn(P\sigma) \cup fn(Q\sigma))\} \cup \{(fn(P\sigma) \cup fn(Q\sigma)) \times \{z\}\}\forall z \in \tilde{z}$

2. ansonsten $\exists Q' : Q\sigma \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}_{D\sigma}Q'$.

$\triangle$

Jedesmal, wenn nun restringierte Namen versendet werden, wird in der Unterscheidung vermerkt, dass diese Namen verschieden von allen freien Namen im übrigen Prozess sind. Die beiden Prozesse aus Gleichung 2.22 sind offen D-bisimular ($P \sim_O^D Q$), da bei Prozess P durch das Versenden des restringierten Namens x in die Unterscheidung die Paare (x, y) und (y, x) aufgenommen werden, d.h. x und y sind in diesem Prozess immer verschieden. Dadurch schlägt der Vergleich im Prozess fehl und P zeigt dasselbe Verhalten wie Q.

Schwache offene D-Bisimulation

Zuletzt wird zu der offenen D-Bisimulation noch eine schwache Version definiert, die, wie bei der schwachen Grund-Bisimulation, von den inneren Aktionen (τ) abstrahiert.

Definition 13 Die *schwache, offene D-Bisimulation* ist eine mit Unterscheidungen indizierte Familie aus einer Menge von symmetrischen, binären Relationen $\mathcal{R}_D$ zwischen Agenten, so dass $\forall\alpha \in Akt$ und $\forall\sigma$, die $\mathcal{D}$ respektieren, gilt:

$$P\mathcal{R}_D Q \wedge P\sigma \xrightarrow{\alpha} P' \wedge bn(\alpha) \cap (fn(P\sigma) \cup fn(Q\sigma)) = \emptyset \Rightarrow$$

1. Wenn $\alpha = \overline{x}\langle(\mathbf{v}\tilde{z}\,)\tilde{y}\rangle$ dann $\exists Q' : Q\sigma \stackrel{\hat{\alpha}}{\Longrightarrow} Q' \wedge P'\mathcal{R}_{D'}Q'$ mit $D' = D\sigma \cup \{\{z\} \times (fn(P\sigma) \cup fn(Q\sigma))\} \cup \{(fn(P\sigma) \cup fn(Q\sigma)) \times \{z\}\}\forall z \in \tilde{z}$

2. ansonsten $\exists Q' : Q\sigma \stackrel{\hat{\alpha}}{\Longrightarrow} Q' \wedge P'\mathcal{R}_{D\sigma}Q'$.

$\triangle$

Sind zwei Prozesse Teil dieser Relation, dann sind sie schwach offen D-bisimular ($P \approx_O^D Q$).

Das Prüfen, ob ein Prozess einen Anderen bisimuliert, kann durch Werkzeuge wie der Mobility Workbench oder ABC (Another Bisimulation Checker) automatisiert erfolgen [57, 13].

Tabelle 2.3: Die Grammatik des π-Kalküls

$$P ::= M \mid P|P' \mid \mathbf{v}z\ P \mid \rho.P \mid A(\tilde{a}) \tag{2.24}$$
$$M ::= 0 \mid \pi.P \mid M + M' \tag{2.25}$$
$$\pi ::= \overline{x}\langle y\rangle \mid x(z) \mid \tau \mid [x = y]\pi \tag{2.26}$$
$$\rho ::= !\overline{x}\langle y\rangle \mid !x(z) \tag{2.27}$$

2.4.5 Der π-Kalkül und Geschäftsprozessmanagement

Die in 2.1 dargestellte Grammatik des π-Kalküls bietet gerade in Bezug auf die angestrebte Automatisierung einen Schwachpunkt. So lassen sich mit dieser Grammatik Prozesse wie $!\tau.0$ ausdrücken, die nach der Semantik der Replikation so oft dupliziert werden müssten, wie sie benötigt werden. Jedoch hat eine Maschine in diesem Fall keine Möglichkeit, die benötigte Anzahl an replizierten Prozessen automatisch festzustellen. Daher wird im π-Kalkül für diese Arbeit die Replikation nur direkt vor einer Kommunikation erlaubt. Das Eintreffen einer Nachricht von einem anderen Prozess oder das erfolgreiche Senden einer Nachricht sind genau definierte Ereignisse, bei deren Eintreten eine Maschine einen Prozess replizieren kann. Daraus ergibt sich die in 2.3 dargestellte neue Grammatik mit der zusätzlichen Regel ρ.

Kapitel 3

Ressourcen und Prozesse

Die Ressourcenorientierung soll helfen, die Skalierbarkeit von Applikationen zu verbessern. Um dies zu veranschaulichen, betrachten wir noch einmal das Beispiel aus der Einführung. Klaus, der Geschäftsführer der Firma ‚XYZ', hat ehrgeizige Ziele und möchte die Zahl der Kunden, die sein Onlineangebot nutzen, in den nächsten zwölf Monaten verdoppeln. Um dieses Ziel zu erreichen, schaltet er Werbung und tritt auf verschiedenen Messen auf. Der Coup gelingt ihm im April: Eine vielgelesene Fachzeitschrift wird auf die Firma aufmerksam und verfasst einen Artikel über die Dienste von XYZ. Kurz nach der Veröffentlichung des Artikels schnellen die Nutzerzahlen in die Höhe. Auf einem Schlag hat Klaus zehnmal mehr Kunden an einem Tag. Das Softwaresystem von XYZ ist hoffnungslos überlastet. Der Administrator besorgt neue, teure Hardware und arbeitet rund um die Uhr. Dennoch sind die Server nur sporadisch zu erreichen. Resigniert kommt der Administrator zu Klaus und sagt: Ich kann machen was ich will, aber unserer System skaliert nicht. Eine Woche später sind die Nutzerzahlen wieder auf ihr normales Niveau gesunken. Im Keller steht jetzt neue, teure aber nicht ausgelastete Hardware.

Obwohl dies wieder eine fiktive Geschichte ist, ist die Problematik jedoch sehr realistisch. Das Phänomen, dass Server nach der Erwähnung eines Dienstes oder einer Web-Seite in einem (Online-)Magazin überlastet sind, ist unter dem Namen *Slashdot-Effekt* berüchtigt (Benannt nach der Nachrichtenseite Slashdot [54]). Aber auch andere Ereignisse können starke Nutzerlast verursachen, unter der schließlich

Server zusammenbrechen. So geschah es zum Beispiel, dass der Online-Fan-Shop eines Fußballvereins nach dessen Aufstieg in die erste Liga wegen Überlastung mehrere Tage nicht benutzbar war.[1] Insbesondere bei kommerziellen Angeboten sind solche Situationen tragisch: Die Server brechen wegen Überlastung zusammen, wenn es am meisten Kunden gibt, wenn also die größten Umsätze erwirtschaftet werden könnten.

Ziel dieses Kapitels soll es sein, Wege aufzuzeigen, wie man mit Hilfe von REST und anderen Technologien Anwendungen skalierbarer gestalten kann. Dazu wird ein Onlineshop vorgestellt, der auf Ressourcen basiert und REST-konform ist. In diesem Zusammenhang werden Ressourcen genau definiert und es wird erklärt, welche Maßnahmen die Skalierung der Anwendung beeinflussen. Desweiteren wird untersucht, wie Prozesse in einer solchen ressourcenorientierten Anwendung eingebettet werden können.

3.1 Wie funktionieren REST-basierte Anwendungen?

Die Frage nach der grundlegende Funktionsweise von REST-basierten Anwendungen soll in diesem Abschnitt anhand einer fiktiven Beispielanwendung in Gestalt eines Onlineshops erläutert werden. Es existieren im Internet eine Unmenge an verschiedenen Onlineshops und die Spanne reicht dabei von handgestrickten Lösungen für kleine Läden bis hin zu globalen Größen wie Amazon, die weltweit bis zu vier Millionen Artikel pro Tag verkaufen [5]. Doch die Strukturen der meisten Onlineshops ähneln sich in dem Punkt, dass ein Warenkorb zur Aufnahme von Artikeln vorhanden ist und sich die Kunden registrieren bzw. anmelden müssen und dass die Shops nicht REST-konform sind. Der hier vorgestellte Onlineshop soll verdeutlichen, wie eine REST-konforme Implementierung aussehen kann. Der grundlegende Vorteil des vorgestellten Onlineshops ist seine gute Skalierbarkeit, die Überlastsituationen der Server vermeiden kann, und dazu beiträgt, extreme Situationen schnell und einfach, zum Beispiel durch die Anmietung weiterer Server, beheben zu können. Es wird angenommen, dass der fiktive Onlineshop über den

[1] Die Seite des Fanshops vom F.C. Hansa-Rostock (`http://www.hansa-fanshop.de`) war nach dessen Aufstieg im Mai 2007 nicht nutzbar. Die Ladezeiten für eine Seite betrugen etliche Minuten.

URI `http://shop.example.com` erreichbar sei. Der Laden bietet eine Reihe von Artikeln an, die hier nicht näher spezifiziert werden.

3.1.1 Artikel Wählen

Über den Server erhält der Client Zugriff auf verschiedene Ressourcen. In einem Onlineshop ist eine Ressource zum Beispiel der Katalog aller angebotener Waren. Die Repräsentation dieses Kataloges, die sich der Client über eine sichere Nachricht (z.B. eine HTTP-GET-Nachricht) abrufen kann, enthält Referenzen auf die verschiedenen Artikel. Jeder Artikel kann als Ressource angesehen werden und der Client ist über die Referenzen im Katalog in der Lage, sich Informationen über die einzelnen Artikel zu beschaffen, indem er eine Repräsentation eines Artikels über eine sichere Nachricht anfragt.

Jede Repräsentation, die der Client vom Server erhält, versetzt den Client in einen neuen Zustand, woher auch der Name Representational State Transfer (repräsentationsbasierte Zustandsüberführung) herrührt. Das wird vor allem bei browserbasierten Anwendungen sichtbar, wo jede Repräsentation direkt gerendert und dargestellt wird. So befindet sich die Applikation anfangs in dem Zustand ‚Katalog wird angezeigt' und wird durch den Aufruf der Referenz zu einem Artikel in den Zustand ‚Artikel wird angezeigt' überführt. Die in einer Repräsentation enthaltenen Referenzen stellen die möglichen Folgezustände der Applikation dar, zwischen denen der Client wählen kann. Dieser Zusammenhang wird durch die Abbildungen 3.1 und 3.2 verdeutlicht. Im ersten Diagramm sind die einzelnen Repräsentationen mit ihren Referenzen auf weitere Ressourcen visualisiert und in Abbildung 3.2 wird das sich daraus ergebene Zustandsmodell der Anwendung auf der Clientseite sichtbar.

Ein Kerngedanke beim Entwurf von REST war die zustandslose Kommunikation. So müssen bei einer Anfrage an den Server alle zur Bearbeitung benötigten Daten mitgesendet werden. Der Server speichert grundsätzlich keine Daten zwischen zwei Anfragen und spart so Systemressourcen, was der Skalierbarkeit des Systems zugutekommt. Die Daten einer Sitzung werden komplett auf der Clientseite verwaltet. Als Nachteil dieser Entwurfsentscheidung kann die Netzwerkverbindung stärker belastet werden, da eventuell Daten mehrfach zwischen Client und

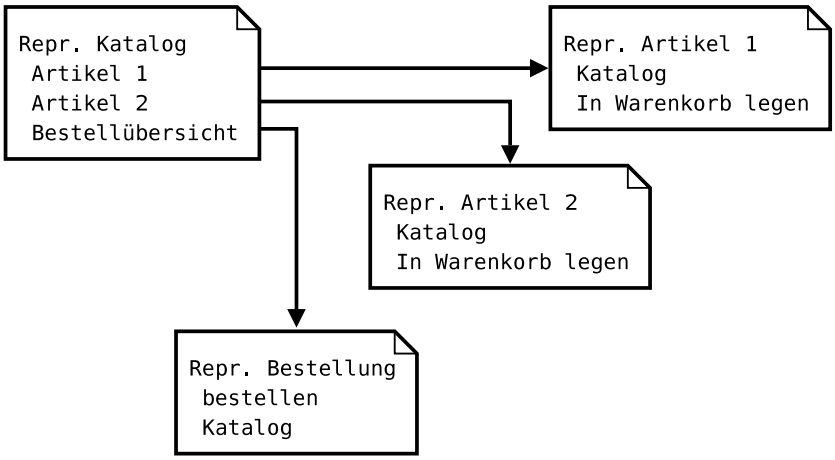

Abbildung 3.1: Repräsentationen mit Referenzen auf Folgezustände

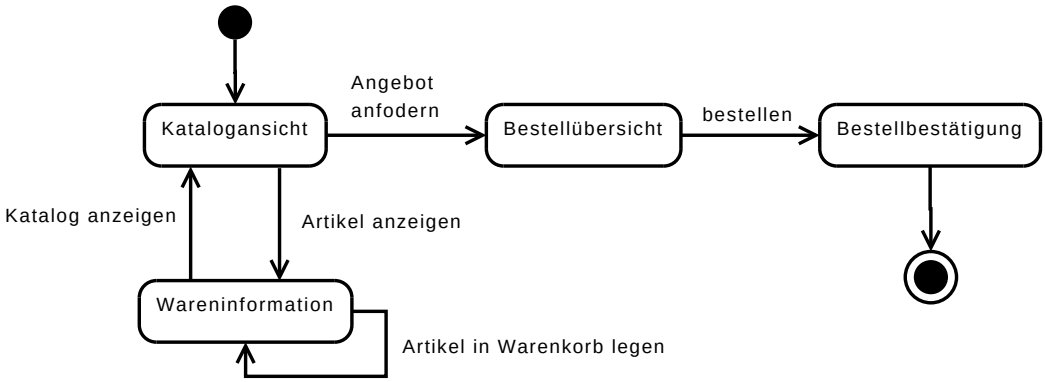

Abbildung 3.2: Zustandsgraph des Clients

Server gesendet werden müssen. Ein Blick auf die heutige Internetstruktur scheint diese Entscheidung dennoch zu rechtfertigen: Überlastsituationen treten wohl eher wegen überlasteter Server auf als durch überfüllte Netze. Gerade die massenhaft angebotenen Shared-Hosting-Pakete tragen zu dieser Situation bei, da sie meist nur über geringe Serverressourcen wie CPU-Zeit oder Speicher verfügen, aber eine schnelle Netzwerkverbindung besitzen.

Viele aktuelle Webanwendungen benutzen heutzutage serverseitige Sitzungsverwaltung, wozu auch der übliche Warenkorb gehört. Sie nutzen also zustandsbehaftete Kommunikation. Zum Teil ist dieser Umstand der fehlenden Verbreitung von Standards für clientseitigen Speicher geschuldet. Es kommen hauptsächlich Cookies zum Einsatz, die in ihrem Funktionsumfang aber sehr eingeschränkt sind. Ein möglicher Kandidat für einen akzeptierten Standard ist Client-side session and persistent storage (SCS) [62], welcher die Speicherung von strukturierten Daten auf der Clientseite ermöglicht. SCS wurde bereits in dem Browser Firefox integriert. Als Alternative bietet beispielsweise Macromedias Flash die Möglichkeit, Nutzerdaten auf dem Client-Rechner zu speichern. Über das Code-on-Demand-Paradigma von REST ist es möglich, Code zur Manipulation dieses Speichers an den Client zu senden (zum Beispiel in Form von JavaScript [29]). Der hier vorgestellte Onlineshop soll zustandslos mit dem Server kommunizieren und alle Sitzungsdaten selber verwalten. Dazu zählt vor allem der Warenkorb. Legt der/die NutzerIn einen Artikel in den Warenkorb, so wird ein Eintrag in den clientseitigen Sitzungsspeicher eingefügt. Es findet dabei keine Kommunikation mit dem Server statt. Soll die Anwendung auf jeder dargestellten Seite Informationen über den Warenkorb präsentieren, etwa den aktuellen Gesamtpreis aller Artikel, so kann ein in der Repräsentation jeder Ressource eingebettetes Skript nach der Übertragung auf der Clientseite die Informationen auslesen und darstellen.

Für das Suchen und Auswählen von Waren sendet der Client nur sichere Anfragen an den Server, deren Antworten durch die zustandslose Kommunikation einfach gecacht werden können. Ein gemeinsamer Cache auf der Serverseite kann die Anfragen nach dem Katalog oder einzelner Artikel für die NutzerInnen schnell beantworten, während ein Cache auf der Clientseite die Repräsentationen von schon einmal aufgerufenen Ressourcen bereit hält. Informationen zur aktuellen Sitzung werden vom Client selber verwaltet und der Server wird damit nicht belastet. Den

Code dazu erhält der Client vom Server. Erst wenn die Waren gekauft werden sollen ‚schiebt die Käuferin ihren Einkaufswagen zur Kasse', d.h. es muss der Inhalt des Warenkorbes an den Server gesendet werden.

Durch die intensive Nutzung von Caches wird die Skalierbarkeit des Systems verbessert. Caches lassen sich vom Prinzip her selber gut skalieren: Übersteigt die Anzahl der Anfragen die Leistungsfähigkeit eines Cache-Servers, können mehrere Caches parallel betrieben werden, wobei jeder Cache-Server dann einen Teil der Anfragen bearbeitet. Übersteigt die zu cachende Datenmenge die Speicherkapazität eines Servers, können mehrere Caches zu Hierarchien zusammengeschaltet werden. Hat ein Server dann die vom Client erfragte Repräsentation nicht gespeichert, gibt er die Anfrage an einen nachgeschalteten Cache der Hierarchie zur Bearbeitung weiter. Für solche Caches gibt es bereits ausgereifte Produkte wie zum Beispiel das quelloffene Squid [55].

Doch ist ein gemeinsamer Cache nur sinnvoll, wenn viele Clients dieselben Anfragen stellen und dieselben Repräsentationen erhalten. Nicht jeder Teil einer Anwendung erfüllt diese Bedingung. Ein Beispiel dafür ist die Bestellabwicklung im Onlineshop, die jetzt genauer betrachtet werden soll.

3.1.2 Bestellvorgang

Der Bestellvorgang unterscheidet sich für jede Kundin und jeden Kunden, da jedes Mal unterschiedliche Artikel bestellt werden. Auch kann die Bestellung nicht vom Client bearbeitet werden. Der Client hat auf die Art und Weise der Abwicklung der Bestellung keinen Einfluss. Angestoßen wird die Bestellung durch eine Nachricht vom Client, in der er mitteilt, welche Waren aus dem Katalog geliefert werden sollen. In der Ressourcenorientierung geschieht dies über eine nicht-idempotente Nachricht. Daraufhin wird ein Bestellvorgang als neue Ressource angelegt und der URI für diese Ressource wird an den Client geschickt. Ruft die Kundin eine Repräsentation dieser Ressource ab, erhält sie den aktuellen Zustand der Bestellabwicklung.

Der Prozess der Bestellabwicklung wird in Abbildung 3.3 in der Business Process Modeling Notation (BPMN) dargestellt. Der Prozess beginnt auf der linken Seite mit einer eingehenden Nachricht, die als Brief symbolisiert ist. Die Nachricht

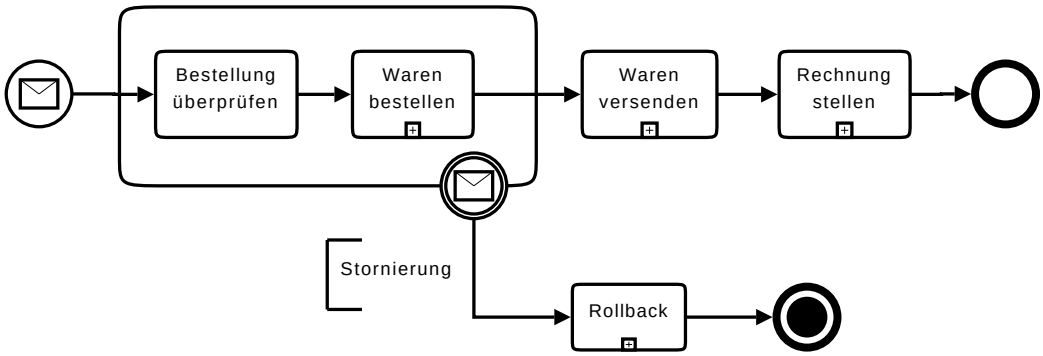

Abbildung 3.3: BPMN-Diagramm der Bestellabwicklung

enthält die Bestellung der Kundin. Der Prozess setzt sich aus mehreren Aktivitäten (abgerundete Rechtecke) zusammen. Die meisten Aktivitäten sind als komplexe Aktivitäten modelliert (wird durch das Plus-Zeichen am unteren Rand einer Aktivität dargestellt), deren Verfeinerungen hier der Einfachheit zuliebe weggelassen wurden. Zuerst wird die Bestellung überprüft und es müssen die Waren beschafft werden. So lange diese beiden Punkte abgearbeitet werden, können die NutzerInnen ihre Bestellungen stornieren. Die Stornierung kann durch eine idempotente Nachricht wie HTTP-DELETE erfolgen, da eine Bestellung nur einmal storniert werden kann und das erneute Senden einer Stornierungsnachricht ein idempotentes Verhalten zeigt. Diese Nachricht wird direkt an die Ressource des entsprechenden Bestellprozesses gesendet. Nach dem Eingang müssen schon getätigte Handlungen, wie die Beschaffung der Waren, rückgängig gemacht werden und der Kundin eventuell eine Stornierungsgebühr in Rechnung gestellt werden. Sobald der Versand der Artikel begonnen hat, kann nicht mehr storniert werden. Zum Schluss muss noch die Rechnung gestellt und die Bezahlung abgewickelt werden.

Nachdem nun erläutert wurde, wie sich ein ressourcenorientierter Onlineshop verhält, soll im weiteren Verlauf dieses Kapitels beschrieben werden, was dieser Ansatz für die Umsetzung von Webanwendungen speziell auf der Serverseite bedeutet. Wie können Ressourcen implementiert werden?

3.2 Implementierung von Ressourcen

In diesem Abschnitt soll auf den Begriff *Ressource* eingegangen und beschrieben werden, wie Server ressourcenorientiert implementiert werden können.

3.2.1 Definition von Ressourcen

Weit gefasst formuliert ist jede Information, die benannt werden kann, eine Ressource. Das könnte ein Bild, ein Dokument oder auch die aktuelle Uhrzeit in Berlin sein. Es existieren statische Ressourcen, deren Repräsentation sich nach dem anlegen der Ressource nicht mehr ändert, aber auch Ressourcen, die zu unterschiedlichen Zeitpunkten andere Repräsentationen besitzen.

Etwas genauer ausgeführt definiert Fielding eine Ressource wie folgt:

> „[...] a resource R is a temporally varying membership function MR(t), which for time t maps to a set of entities, or values, which are equivalent. The values in the set may be resource representations and/or resource identifiers.“ [20]

Über eine Ressource werden so ein oder mehrere Bezeichner einer oder mehreren Entitäten und ihren Repräsentationen zugeordnet. Diese Zuordnung kann sich über die Zeit verändern. Als Gründe für diese abstrakte Definition wird angeführt, dass durch die Generalität des Ansatzes viele verschiedenartige Informationsquellen einbezogen werden können und dass durch das späte Binden von Referenzen zu Repräsentationen Content-Negotiation auf Grundlage von einzelnen Anfragen stattfinden kann. Außerdem wird so ermöglicht, Konzepte statt einzelne Repräsentationen zu referenzieren. Ändert sich die Repräsentation einer Ressource, müssen die Referenzen, die auf diese Ressource zeigen, nicht geändert werden. Entscheidend ist aber, dass sich die Semantik einer Ressource über die Zeit nicht ändert.

Möchte ein Poet seine Werke online veröffentlichen, könnte er sein neuestes Gedicht unter dem URI http://example.com/mein_neuestes_gedicht veröffentlichen. Die Repräsentation dieser Ressource enthält den Text dieses Gedichtes. Schreibt er ein weiteres Gedicht, ist es unter dem selben URI verfügbar. Das Konzept hinter der Ressource (sein neuestes Gedicht) und der URI sind gleich geblieben, geändert

wurde nur die Repräsentation. Durch Content-Negotiation handelt der Client mit dem Server ein gewünschtes Format für den Inhalt einer Ressource aus. So könnte das Gedicht in einer PDF-, HTML- oder Text-Repräsentation ausgeliefert werden.

Wird diese Definition von Ressourcen in Bezug auf die Client-Server-Architektur von REST betrachtet, ergeben sich zwei Sichtweisen. Der Client kennt ein URI (http://shop.exampel.com), der eine Ressource referenziert und schickt er eine Anfrage an diesen URI, erhält er eine aktuelle Repräsentation der Ressource (z.B. den Warenkatalog). Der Server kennt ‚abstrakte' Ressourcen und weiß, welche URIs welche Ressourcen identifizieren. Außerdem benötigt der Server für jede Ressource eine Repräsentation oder eine *Implementierung*, welche die Repräsentation liefert. Es können mehrere Ressourcen dieselbe Repräsentation oder Implementierung besitzen.

Um den Unterschied zwischen einer Ressource und ihrer Implementierung deutlich zu machen, sollen die beiden Begriffe hier getrennt voneinander definiert werden.

Definition 14 Eine *Ressource* ist ein Konzept, das über einen oder mehreren URIs referenziert werden kann und dem über die Zeit verschiedene Repräsentationen zugeordnet werden können. △

Wenn es sich bei der aktuellen Repräsentation der Ressource nicht um ein Textdokument, ein Bild oder eine binäre Datei handelt, liegt sie oftmals nicht explizit auf dem Server vor, sondern wird durch ein Programm zum Beispiel mit Daten aus einer Datenbank dynamisch erzeugt. Der Server muss einer Ressource in diesem Fall das Programm zuordnen, welches die Repräsentation erstellt.

Definition 15 Die *Implementierung einer Ressource* ist ein Programm, mit dessen Hilfe der Server auf eine Anfrage eine Repräsentation der der Implementierung zugeordneten Ressource erstellen kann, bzw. eine übermittelte Repräsentation behandeln kann. △

In Bezug auf das Beispiel des Onlineshops könnte jeder angebotene Artikel einem Datensatz in einer Datenbank entsprechen. Ruft der Client ein URI auf, der einem Artikel zugeordnet ist, kann die zugehörige Implementierung den Datensatz aus der Datenbank lesen und eine für den Client geeignete Repräsentation erstellen, beispielsweise eine HTML-Seite mit der Artikelbeschreibung.

3.3 Portable Ressourcen

Im Onlineshopbeispiel ist der erste Teil der Anwendung, der die Artikel und den Katalog umfasst, durch die zustandslose Kommunikation und die Cachebarkeit aller Repräsentationen sehr skalierbar ausgelegt. Doch wie können die nicht-cachebaren Teile der Anwendung skalieren?

Der Gefahr einer Überlastsituation einer Anwendung kann durch zwei verschiedene Varianten begegnet werden: Entweder, man tauscht die aktuelle Hardware gegen eine potentere Maschine aus (Scaling-Up) oder man verteilt die Anwendung auf mehrere Maschinen (Scaling-Out). Der erste Ansatz hat den Vorteil, dass dies bei vielen Softwaresystemen schnell zum Erfolg führt. Er ist aber von der Sache her nach oben beschränkt: Steht bereits die schnellste passende Hardware, für die viel Geld zu zahlen war, im Serverraum, hat man kaum noch Handlungsmöglichkeiten. Das Scaling-Out-Konzept muss von der Software explizit unterstützt werden. Anwendungsteile müssen gegebenenfalls über Rechnergrenzen hinweg miteinander kommunizieren. Dafür ist der Skalierung der Anwendung kaum Grenzen gesetzt. Treten Performanz-Probleme auf, wird einfach ein neuer Server hinzugekauft oder angemietet. Für kurzfristige Überlastsituationen bietet es sich auch an, weitere Server in fremden Rechenzentren stundenweise zu mieten.

Die Aufteilung von Anwendungen in einzelne Ressourcen bietet die Möglichkeit, ein Scaling-Out-Konzept zu verwenden. Wenn es gelingt, einzelne Ressourcen von einem Rechner auf einen anderen zu verschieben, könnte man so Server entlasten, die gerade besonders stark beansprucht werden. Die Ressourcen werden *portabel*.

Wird eine Ressource von einem Server auf einen anderen verschoben, muss auch die Implementierung der Ressource mit umziehen. Die Infrastruktur muss über den Wechsel informiert werden, damit Anfragen von Clients an diese Ressource zum neuen Server geleitet werden können.

Weil mehrere Ressourcen ein und dieselbe Implementierung besitzen können, zieht genaugenommen die Implementierung einer Ressource um und nimmt alle sie benutzenden Ressourcen mit. Im Beispiel könnten die beiden Konzepte *Bestellvorgang 123* und *Mein aktueller Bestellvorgang* zu einem bestimmten Zeitpunkt dieselbe Implementierung verwenden, nämlich genau dann, wenn mein aktueller

Bestellvorgang der Bestellvorgang 123 ist. Wird die Implementierung des Vorganges (der zu Grunde liegende Prozess) verlagert, müssen auch beide Ressourcen mit verlagert werden. Es muss also auch die Information gepflegt werden, von welchen Ressourcen eine Implementierung genutzt wird.

Im Umkehrschluss bedeutet dies aber auch, dass eine Ressource eventuell den Server wechseln muss, wenn sie eine andere Implementierung wählt. Ist der Bestellvorgang 123 abgeschlossen und der Vorgang 124 wird gestartet, dann erhält die Ressource *Mein aktueller Bestellvorgang* einen neuen Prozess als Implementierung, der gegebenenfalls auf einem anderen Server ausgeführt wird. Die Ressource muss (diesmal ohne Implementierung) umziehen, d.h. ihre URIs müssen auf die neue Implementierung abgebildet und die Routinginformationen in der Infrastruktur müssen angepasst werden, um alle Anfragen korrekt weiterleiten zu können. In Abbildung 3.4 wird gezeigt, wie Ressourcen und ihre Implementierungen auf verschiedene Server verteilt wurden. Die Infrastruktur weiß, welche Ressource auf welchem Server existiert und kann die Anfragen von Clients entsprechend Umleiten.

Doch damit die Implementierungen portabel werden, müssen einige Voraussetzungen erfüllt sein. Als erstes stellt sich die Frage, wie Ressourcenimplementierungen voneinander abzugrenzen sind. Zur Abgrenzung von einzelnen Implementierungen hilft das Konzept des transaktionalen Geltungsbereichs. Bei verteilten Anwendungen entsteht das Problem, dass über mehrere Rechner verteilte Daten innerhalb einer Transaktion geändert werden müssen. Dazu gibt es einige Ansätze für *verteilte Transaktionen*, zum Beispiel 2PC oder Paxos [31]. Helland bemerkt jedoch, dass der Einsatz von verteilten Transaktionen sehr komplex sei und viele Projekte an Performanzproblemen gescheitert seien [25]. Für den Entwurf von skalierbaren, verteilten Anwendungen sind verteilte Transaktionen nicht erforderlich, wenn in einer Transaktion nur Daten auf einem Rechner manipuliert werden. Lokal auf einer Maschine stellt die Umsetzung von Transaktionen kein Problem dar. Es gilt also, den Onlineshop so zu entwerfen, dass Transaktionen nicht mehrere Server umspannen.

Doch was sind Transaktionen in der Ressourcenorientierung? REST nutzt ausschließlich zustandslose Kommunikation. Das heißt, es besteht auf der Serverseite kein Bezug zwischen zwei Anfragen. Eine Transaktion im Sinne von REST kann

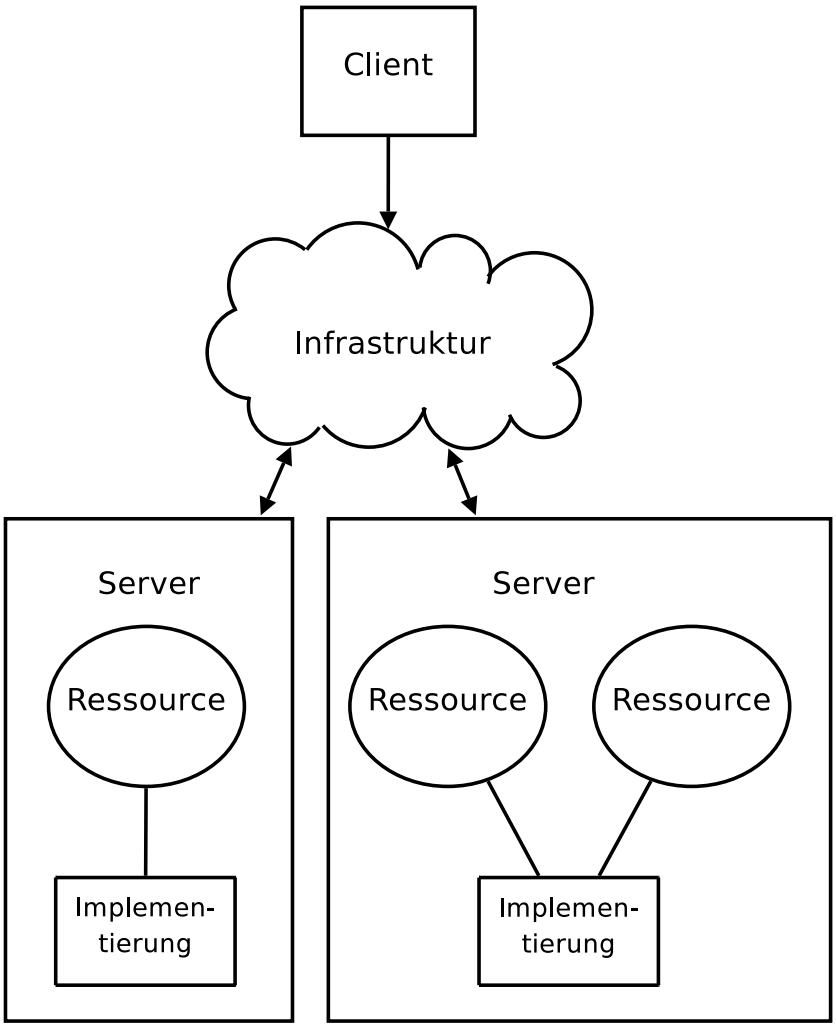

Abbildung 3.4: Verteilung von Ressourcen und ihren Implementierungen über mehrere Server

demnach nicht mehrere Interaktionen (Anfrage/Antwort-Zyklen) umfassen. Auf der anderen Seite ist der Client, der eine Anfrage an den Server schickt, nicht für Fehler auf dem Server verantwortlich. Sendet der/die NutzerIn eine Bestellung an den Server und tritt bei der Bearbeitung ein Fehler auf, so antwortet der Server beispielsweise mit dem HTTP-Fehler `500 Internal Server Error`. Für den/die NutzerIn ist in diesem Fall klar, dass die Bestellung nicht erfolgt ist. Der Server muss im Fehlerfall dafür Sorge tragen, dass bereits vorgenommene Datenänderungen rückgängig gemacht werden. Die Bearbeitung einer Anfrage auf dem Server muss also atomar erfolgen. Als Schlussfolgerung daraus entspricht eine Transaktion in der Ressourcenorientierung genau einer Interaktion, da nicht mehrere Interaktionen zu einer Transaktion zusammengefasst werden können und jede Interaktion transaktionales Verhalten zeigt.

Soll auf verteilte Transaktionen verzichtet werden, müssen alle Daten, die bei einer Interaktion benutzt werden, auf der Maschine vorliegen, auf der die Implementierung ausgeführt wird. Zieht die Implementierung auf einen anderen Server, müssen all ihre Daten mittransferiert werden. Da die einzelnen Ressourcenimplementierungen unabhängig voneinander verlagert werden sollen, ist es zwingend erforderlich, dass die Daten aller Implementierungen paarweise disjunkt sind. Ein Datum gehört zu genau einer Ressourcenimplementierung. Wird bei zwei Interaktionen auf dieselben Daten zugegriffen, folgt daraus, dass beide Interaktionen dieselbe Implementierung verwenden.

3.4 Prozessbasierte Implementierung

Derzeit werden viele Webanwendungen mit Skriptsprachen wie Perl, PHP oder Ruby implementiert. Der Server leitet dazu über eine Schnittstelle wie CGI die Anfragen an die Skripte weiter, wo sie bearbeitet und Antworten erstellt werden. Derzeitiger Stand der Technik ist die Entwicklung solcher Applikationen mit Hilfe von Webframeworks, die den Aufwand der Programmierung deutlich reduzieren können. Als Beispiele seien hier Ruby On Rails, ASP.NET und Java Server Faces genannt. [51, 33, 56] Wird jedoch das Ziel verfolgt, Geschäftsprozesse ressourcenorientiert zu implementieren, stößt man mit dieser Art von Webanwendungen schnell an dieselben Grenzen, die auch bei nicht weborientierter Unternehmenssoftware

existieren. Durch die fest einprogrammierte Reihenfolge der möglichen Anwendungszustände ist es nur mit hohem Aufwand möglich, Änderungen im zu Grunde liegenden Geschäftsprozess vorzunehmen. Die Flexibilität der betriebswirtschaftlichen Prozesse wird so durch die Unflexibilität der IT-Infrastruktur eingeschränkt.

Um diesen Problemen bei Unternehmenssoftware zu begegnen, wurden verschiedene Ansätze entwickelt. In der Serviceorientierung wird versucht, die Software in der Art zu modularisieren, dass einzelne abgeschlossene Funktionsblöcke als Dienste gekapselt werden. Anwendungen können so durch die Aneinanderreihung von Dienstaufrufen realisiert werden. Der Prozess, der die Dienstaufrufe verbindet, kann unter Berücksichtigung der vorhandenen Datenflüsse relativ einfach umgestellt oder erweitert werden. Hinzu kamen Überlegungen, ähnlich wie beim Model Driven Development (MDD), einen modellgetriebenen Entwicklungsprozess auch bei prozessbasierten Anwendungen zu verwenden, vorrangig um Kosten bei der Entwicklung und Wartung einzusparen. Der Kerngedanke dabei ist, dass ein Domänenexperte, etwa ein Betriebswirt oder eine Betriebswirtin, den Prozess in einer Notation wie BPMN modelliert und die Umsetzung des Prozesses in eine Anwendung zum größten Teil automatisiert erfolgt. Dieser Gedanke wurde zum Beispiel beim Forschungsprojekt Process Family Engineering in Service-Oriented Applications (PESOA) mit dem Fokus auf Prozessfamilien umgesetzt [60]. In den beiden Anwendungsgebieten Automotive und E-Business sollten Produktfamilien durch einen Prozess mit mehreren Variationspunkten modellgetrieben entwickelt und gewartet werden. In diesem Rahmen gelang es dem Projekt Magrathea [18] eine Webanwendung (ein Hotelbuchungsportal) aus einem Prozess automatisiert zu generieren. Wurde der Prozess umgestellt oder wurden andere Alternativen bei den Variationspunkten gewählt, konnten diese Änderungen automatisiert in die Anwendung übernommen bzw. weitere Mitglieder der Produktfamilie generiert werden.

Im Unterschied zu solchen generativen Ansätzen, bei denen modellgetrieben herkömmliche Anwendungen erzeugt werden, nutzen interpretative Techniken eine Prozessbeschreibung direkt als Implementierung. Die Prozessbeschreibung wird erst zur Laufzeit interpretiert. Dieses Vorgehen bringt unter anderem Vorteile beim Monitoring, da zur Laufzeit der Zusammenhang zwischen der gerade ausgeführten Ausprägung des Prozesses und der Prozessbeschreibung explizit vorliegt. Es ist zu

jedem Zeitpunkt eindeutig, in welchen (Prozess-)Zustand sich die Anwendung gerade befindet. Weitere Vereinfachungen ergeben sich beim Deployment: Es genügt eine Prozessbeschreibung an eine Ausführungsmaschine zu senden, der Schritt der Generierung entfällt vollständig. Jedoch wird immer eine Ausführungsmaschine benötigt, die die Prozesse interpretiert, was auch zu einer erhöhten Ausführungszeit führen kann.

Für die Interpretation von Geschäftsprozessen hat sich der Standard BPEL etabliert [6]. Die Prozesse müssen zunächst in das von BPEL verwendete XML-Format überführt werden. In der Spezifikation von BPMN [63] ist eine Abbildungsvorschrift enthalten, die die Überführung von BPMN-Modellen in das BPEL-Format beschreibt. BPEL ist ausschließlich dazu in der Lage, Web-Services im Sinne der WSDL-Spezifikation [15] aufzurufen und beliebig zu kombinieren und selber Web-Services anzubieten.

Aber auch für verteilte prozessbasierte Geschäftsanwendungen scheint es interessant, REST als Architekturstil zu benutzen, um so eine besonders hohe Skalierbarkeit zu erreichen. In der Ressourcenorientierung könnten Prozesse als Implementierungen von Ressourcen dienen, d.h. wird eine Repräsentation einer Ressource vom Client angefordert, so wird ein Prozess angestoßen, der zum Beispiel Daten aufbereitet und diese als Repräsentation zurückgibt. Im Onlineshop-Beispiel wird der Prozess der Bestellabwicklung als Ressource dargestellt und bekommt einen eigenen URI, über den der/die NutzerIn den aktuellen Zustand der Bestellabwicklung abfragen kann.

Durch die formale Verifizierbarkeit erscheint, wie in Kapitel 2.4 dargelegt, der π-Kalkül als ein geeigneter Kandidat, Geschäftsprozesse automatisiert auszuführen. In dieser Arbeit soll diese These durch die Implementierung eines prototypischen Ressourcenservers überprüft werden, der die Realisierung von Ressourcen durch die Interpretation von π-Prozessen ermöglicht.

3.5 Zusammenfassung

In diesem Kapitel wurde zuerst ein Beispiel eines REST-konformen Onlineshops eingeführt und es wurde gezeigt, wie die Umsetzung der Merkmale dieses Architekturstils die Skalierbarkeit der Anwendung verbessern kann. Der Begriff der Res-

source wurde genau definiert und es wurde gezeigt, wie portable Ressourcenimplementierungen die Skalierbarkeit von nicht zu cachenden Anwendungsteilen steigert. Zum Schluss wurden die Vorteile einer Implementierung hervorgehoben, die direkt durch die Interpretation von Prozessen zur Laufzeit realisiert wird.

Für die Darstellung und Interpretation von Geschäftsprozessen im Internet bietet sich der π-Kalkül an, da sich mit ihm, wie in Kapitel 2.4 beschrieben, agile Prozesse modellieren lassen, deren Umwelt sich durch die Kommunikation von Referenzen verändern kann. Doch in einigen Punkten weicht die Theorie des π-Kalküls stark von den Gegebenheiten im Internet ab. Deswegen soll im nächsten Kapitel geklärt werden, ob der π-Kalkül für den Einsatz in einer ressourcenorientierten Umgebung geeignet ist.

Kapitel 4

Der π-Kalkül im Internet

Der π-Kalkül erlaubt die formale Verifikation von Prozessen und kann durch die Möglichkeit der Link-Passing-Mobility auch agile Prozesse beschreiben, deren einzelne Beteiligten erst im Laufe des Prozesses bekannt werden, was die Modellierung von typischen, service-orientierten Architekturen im Internet erlaubt. Doch können π-Modelle nicht nur zur theoretischen Beschreibung sondern auch als Implementierung von realen Prozessen genutzt werden? Lässt sich die formale Kommunikation von π-Prozessen auf existierende Medien wie dem Internet übertragen?

Für den Einsatz im Internet führt kaum ein Weg an etablierten Protokollen wie HTTP, Transmission Control Protocol (TCP) [46] und Internet Protocol (IP) [45] vorbei, vor allem dann nicht, wenn eine Integration mit bestehenden Systemen geplant ist. Doch nutzen diese Protokolle andere Konzepte als der π-Kalkül, ja sie sind teilweise sogar gegensätzlich. So geht der π-Kalkül von einem verbindungslosen, blockierenden Kommunikationsverhalten aus, während TCP verbindungsorientiert arbeitet und ein Timeout blockierende Verbindungen verhindert. Für die Nutzung von HTTP ist die Verwendung von URIs unumgänglich, jedoch enthalten sie zum Teil im Gegensatz zu π-Namen Lokalisierungsinformationen, die ihre Benutzung im π einschränken würden.

Als Architekturstil empfiehlt sich, wie im vorherigen Kapitel dargestellt, der Representational State Transfer. Könnte der π-Kalkül in diesem Zusammenhang helfen, die Einhaltung der festgelegten Semantik der einheitlichen Schnittstelle von

REST formal zu verifizieren?

Ist der π-Kalkül überhaupt für die Implementierung von Prozessen im Internet tauglich? Existieren Möglichkeiten, die genannten Hürden zu überwinden? Das soll Thema dieses Kapitels sein. Außerdem soll erörtert werden, wie Funktionalität, die nicht in π formuliert ist, in π-Prozesse eingebunden werden kann.

4.1 Architektur

Um die Konzepte, die in diesem Kapitel eingeführt werden, besser erklären zu können, soll hier zuerst die Architektur für einen π-basierten Ressourcenserver (kurz *π-Server*) vorgestellt werden. Der π-Server besteht, wie in Abbildung 4.1 ersichtlich, aus zwei Akteuren, dem *HTTP-Server* und der *virtuellen Maschine* (VM). Da REST als Architekturstil verwendet werden soll, ist die Kommunikation als Client-Server-Kommunikation ausgelegt. Eine Kommunikation wird ausschließlich vom Client angestoßen. Jedoch können Prozesse in beiden Rollen tätig sein. Dies ist zum Beispiel bei der Dienstkomposition notwendig: Ein Prozess erhält in der Serverrolle eine Anfrage eines Clients und schickt anschließend selber Anfragen an andere Server, um deren Antworten zu komponieren.

Der generelle Ablauf des Servers sieht wie folgt aus: Erhält der π-Server eine Anfrage eines Clients, so wird diese zuerst vom HTTP-Server angenommen. Steht ein Prozess bereit, der diese Anfrage bearbeiten kann, so wird sie an die virtuelle Maschine weitergereicht. Die virtuelle Maschine hat Zugriff auf alle π-Prozessbeschreibungen (im Folgendem als *π-Prozesse* bezeichnet) und führt die entsprechenden π-Prozesse Schritt für Schritt aus. Enthält eine Prozessbeschreibung einen Zugriff auf externe Ressourcen, startet die virtuelle Maschine eine HTTP-Kommunikation mit dem entsprechenden Server und führt die erhaltene Antwort dem Prozess zu.

Im nächsten Abschnitt wird gezeigt, wie die Kommunikation in beiden Rollen im Detail aussieht.

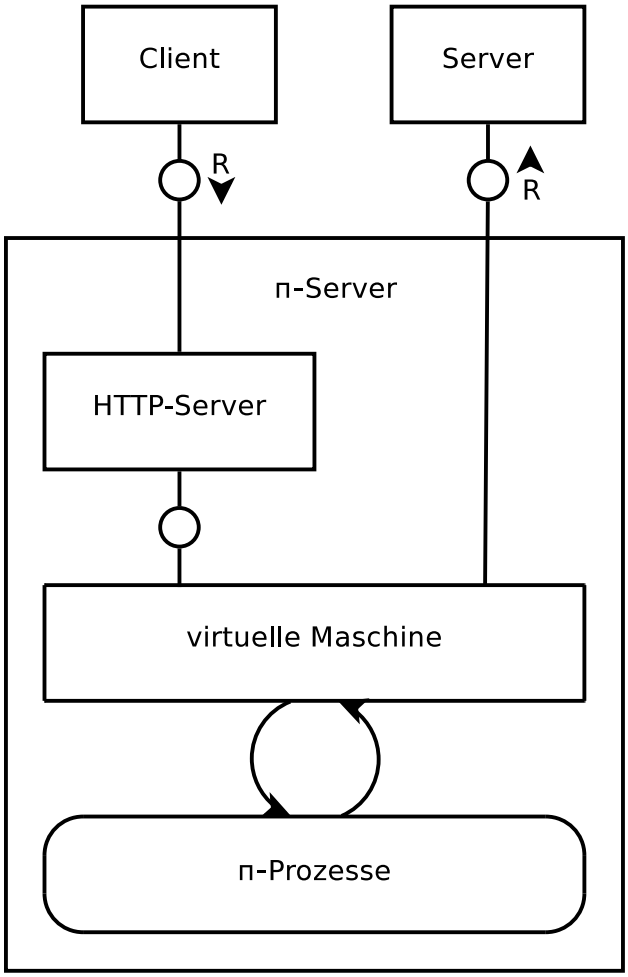

Abbildung 4.1: Architekturübersicht des Systems

4.2 Kommunikation

Für die Kommunikation im World Wide Web werden hauptsächlich die Protokolle HTTP [21] und TCP/IP [46, 45] verwendet. Kommunikationsendpunkte werden durch URIs angegeben. Dieser Abschnitt beschreibt, wie diese Konzepte in einem π-Server umgesetzt werden können, sodass die Semantik des π-Kalküls nicht verletzt wird. Zuerst soll die Verwendung von URIs beschrieben werden. Anschließend wird die Verbindungsorientierung und das blockierende Kommunikationsverhalten des π-Kalküls untersucht.

4.2.1 Uniform Resource Identifiers (URI)

Prozesse im π-Kalkül kommunizieren über Namen während im Internet URIs dafür verwendet werden. Da die π-Namen ein sehr weitgreifendes Konzept darstellen, können auch URIs als Namen verwendet und so eine Kommunikation von π-Prozessen über das Internet ermöglicht werden. Wie das im π-Kalkül aussieht, soll an der fiktiven Firma ‚Example' gezeigt werden, die den als π-Prozess realisierten Dienst ‚service' anbietet.

$$Service(broker_{register}) \stackrel{def}{=} \mathbf{v} http://example.com/service\ .$$

$$\overline{broker_{register}}\langle http://example.com/service\rangle.$$

$$!http://example.com/service(request).P$$

Beim Start wird dem Prozess der URI eines Service-Brokers mitgegeben, bei dem sich der Dienst registrieren kann, um von Clients gefunden zu werden. Anschließend wird der Name des Dienstes als URI erzeugt und über den Broker veröffentlicht. Danach wartet der Dienst auf eingehende Anfragen, in dem er von dem URI liest. Kommt eine Anfrage herein, wird der Prozess P gestartet, um die Anfrage zu bearbeiten.

Es sind jedoch einige Unterschiede zwischen π-Namen und URIs zu erörtern. Im π-Kalkül können Namen durch den Restriktionsoperator $\mathbf{v}$ auf einen Bereich beschränkt werden. Restringierte Namen sind per Definition verschieden von allen anderen Namen. Dennoch können gleiche Bezeichner für verschiedene Na-

men genutzt werden, etwa wenn zwei Prozesse P und Q unabhängig voneinander den Namen a restringieren. Das a von P ist dann verschieden vom a von Q. Erfolgt eine Scope-Intrusion (P sendet sein a an Q), müssen gegebenfalls Namenskonflikte durch α-Konvertierung (Umbenennung) aufgelöst werden. Im Gegensatz dazu sind URIs (abgesehen von den URIs, die sich auf den aktuelle Host beziehen wie `http://localhost` oder `http://127.0.0.1`) zu jedem Zeitpunkt systemweit eindeutig. Sind zwei URIs identisch, so bezeichnen sie auch dieselbe Ressource. Die Restriktion von Namen muss demnach besonders behandelt werden, wenn die Namen URIs darstellen.

Im Internet wird die eindeutige Vergabe von URIs durch *Namensautoritäten* gewährleistet [37]. Dazu enthalten die meisten URIs, unter anderem die im HTTP-Schema, den Domainnamen und davon ausgehend einen Pfad innerhalb dieser Domain. Die Domainnamen werden durch Registrare zentral verwaltet (zum Beispiel von der DENIC [17] für alle .de-Domains). Der Inhaber einer Domain oder Subdomain besitzt die Namensautorität für diesen Bereich und kann bzw. muss sich um die Eindeutigkeit der URIs in seinem Bereich kümmern. Für die Implementierung der virtuellen Maschine bedeutet dies, dass der Restriktionsoperator **v** die Eindeutigkeit des Namens prüfen muss. Ist der URI bereits vergeben, kann eine noch freie URI gewählt und im π-Prozess durch α-Konvertierung eingefügt werden. Wäre im Beispiel der URI `http://example.com/service` schon einer anderen Ressource zugeordnet, könnte er im π-Prozess durch `http://example.com/service1` ersetzt werden. Im Großen und Ganzen ähnelt das Vorgehen dem des π-Kalküls, nur dass Namenskonflikte bereits bei der ‚Erzeugung' der Namen, und nicht erst bei Scope-Intrusion aufgelöst werden.

Bei der Kommunikation über HTTP enthalten die URIs, wie gerade erwähnt, Informationen zur Domain bzw. zum Host und einem Pfad innerhalb dieser Domain. Sie sind so nicht nur ein eindeutiger Bezeichner einer Ressource sondern geben auch an, wie man sie im Internet finden kann. Im Beispiel ist bei dem URI `http://example.com/service` example.com der Host, zu dem Anfragen an den URI geleitet werden, und /service der Pfad auf diesem Server. Dies schränkt die Flexibilität von Namen insoweit ein, dass nur π-Prozesse von einem URI lesen können, wenn der URI auf den Host zeigt, auf dem der Prozess ausgeführt wird. Ein π-Prozess auf dem Host example.com wäre nicht in der Lage, von der URI

google.com zu lesen, da alle Anfragen an google.com durch die Infrastruktur an die Server von Google geleitet werden (Die Möglichkeit, dass zwei verschiedene Adressen auf ein und denselben Host abgebildet werden können, wird hier nicht weiter berücksichtigt). Alle π-Prozesse, die mit URIs arbeiten, müssen diese Einschränkung berücksichtigen, sonst kann es zu Deadlocks kommen. Eine Implementierung kann die Erreichbarkeit von URIs sicherstellen, indem über den Restriktionsoperator nur URIs erzeugt werden, für die der Betreiber die Namensautorität besitzt und nur von selbsterstellten URIs gelesen wird.

4.2.2 Verbindungsorientierte Kommunikation

REST geht von einer Client-Server-Kommunikation aus, das heißt Kommunikationen werden ausschließlich vom Client angestoßen. Dazu schickt dieser eine Anfrage an den Server, welcher daraufhin eine Antwort zurücksendet. Durch die Verwendung des verbindungsorientierten Transportprotokolls TCP werden Anfrage und Antwort über dieselbe, vom Client initiierte, Verbindung gesendet. Da der π-Kalkül keine Möglichkeit vorsieht, verbindungsorientierte Kommunikation explizit darzustellen, kann diese nicht im π-Prozess selber beschrieben werden. So muss eine andere Komponente, zum Beispiel der HTTP-Server, dafür Sorge tragen, dass die Antwort vom π-Prozess über die richtige Verbindung zum Client geschickt wird.

Prinzipiell nimmt der HTTP-Server eine Anfrage von einem Client entgegen und leitet sie über die virtuelle Maschine an den π-Prozess weiter, der auf den URI lauscht, an den die Nachricht geschickt wurde. Kümmert sich der HTTP-Server um die Verbindung zum Client, muss die Antwort, die der π-Prozess generiert, an den HTTP-Server (siehe Abbildung 4.1) zurück gehen. Der HTTP-Server hält für die Zeit der Bearbeitung die Verbindung zum Client offen und schickt die Antwort vom π-Prozess über diese Verbindung zum Client.

Im Detail sieht es so aus, dass der HTTP-Server einen eindeutigen π-Namen o erstellt (bzw. restringiert) und diesen zusammen mit der Anfrage an den π-Prozess P übergibt. Nach der Bearbeitung nutzt P den Namen o als Kanal, um die generierte Antwort an den HTTP-Server zu leiten. Dabei kann der Name o bei der Prozessbearbeitung auch an andere π-Prozesse kommuniziert werden, es muss nur sichergestellt sein, dass genau eine Antwort an den HTTP-Server zurückgesendet

wird. Formal lässt sich dieses Verhalten eines π-Prozesses P prüfen, indem gezeigt wird, dass der Prozess P den folgenden Prozess Q schwach offen D-bisimuliert, $P \approx_O^D Q$.

$$Q \stackrel{def}{=} uri(request, o).\tau.\overline{o}\langle resp\rangle \tag{4.1}$$

4.2.3 Blockierende Kommunikation

Im Gegensatz zu üblichen Kommunikationssystemen, bei denen Zugriffsversuche auf nicht vorhandene Quellen fehlschlagen, besitzt der π-Kalkül ein blockierendes Kommunikationsverhalten, das heißt, dass der Lese- oder Schreibversuch auf einen Namen solange blockiert, bis ein parallel laufender Prozess auf denselben Namen zu schreiben bzw. zu lesen versucht.

Sollen π-Prozesse in bestehende Systeme, wie dem Internet, integriert werden, muss die Implementierung einer Ausführungsmaschine für den π-Kalkül Vorkehrungen treffen, um zum Einen die Erwartungen anderer NutzerInnen und Systeme an das Verhalten von Ressourcen im Netz zu entsprechen und zum Anderen die Regeln des π-Kalküls korrekt umzusetzen. Andernfalls wären theoretische Erkenntnisse über π-Prozesse nicht auf die Implementierung übertragbar.

Um eine Lösung für dieses Problem zu finden, kann man das Kommunikationsverhalten in den einzelnen Rollen der π-Prozesse als Server und als Client untersuchen.

Wird das Serververhalten betrachtet, so versucht der Prozess von einem Kanal, der ein URI darstellen kann, zu lesen. Der Leseversuch eines π-Prozesses blockiert, bis ein Client eine Anfrage auf diesen Kanal sendet. Das entspricht dem typischen Verhalten von Servern.

Für die Entgegennahme der Anfragen der Clients ist jedoch vor den eigentlichen Prozessen ein HTTP-Server verantwortlich. Dieser verwaltet die Verbindungen zu den Clients und leitet Anfragen an die entsprechenden π-Prozesse weiter. Sendet ein Client jedoch eine Anfrage an ein URI, der von keinem Prozess bearbeitet wird, muss der HTTP-Server eingreifen. Im Sinne des π-Kalküls müsste der Server die Verbindung nun so lange aufrecht erhalten, bis ein π-Prozess versucht, von dem entsprechenden URI zu lesen. Dies ist jedoch durch den Verbindungs-

Timeout von TCP technisch nicht möglich und erscheint auch sonst nicht vorteilhaft. Das Halten von Verbindungen für nicht vorhandene Ressourcen beansprucht Systemressourcen auf dem Server. Außerdem erwarten die meisten Clients eine Fehlermeldung, wenn die gewünschte Ressource momentan nicht vorhanden ist, um auf die Situation angemessen reagieren zu können. Deswegen sollte der HTTP-Server auf eine Anfrage an eine nichtvorhandene Ressource mit der HTTP-Fehlermeldung ‚404 - File not found' antworten.

Tritt ein π-Prozess hingegen als Client auf, so kann es geschehen, dass die von ihm aufgerufene Ressource nicht vorhanden ist und er wird die Fehlermeldung ‚404 - File not found' erhalten. Dies widerspricht der blockierenden Semantik des π-Kalküls und wäre ein Bruch mit der Theorie. Um dennoch ein blockierendes Verhalten zu gewährleisten, muss hier die virtuelle Maschine intervenieren, die die Kommunikation der π-Prozesse ausführt. Enthält ein π-Prozess einen Schreibversuch auf einen URI, so startet die virtuelle Maschine eine HTTP-Anfrage. Erhält sie daraufhin den Fehlercode 404, sendet sie die Anfrage erneut und zwar so lange, bis die gewünschte Ressource vorhanden ist. Um Systemressourcen zu sparen, sollte zwischen zwei Anfrageversuchen eine Wartezeit eingeschoben werden.

Besondere Behandlung benötigt der Fall, bei dem die Kommunikation innerhalb einer Summation vorkommt:

$$P(a, b, c) \stackrel{def}{=} \overline{a}.Q + \overline{b}.R + \overline{c}.S \tag{4.2}$$

Die π-Semantik besagt hier, das derjenige Pfad der Summation exklusiv gewählt wird, dessen Schreibversuch als erstes gelingt. In der Theorie ist nicht festgelegt, welcher Pfad gewählt wird, wenn mehr als ein Schreibvorgang gleichzeitig möglich wäre. Ein Algorithmus für das Vorgehen der virtuellen Maschine könnte wie folgt aussehen: Versuche, eine Anfrage an a zu senden. Gelingt dies, mache weiter als Q. Ist die Ressource, die a bezeichnet, nicht verfügbar, sende Anfrage an b. Ist b auch nicht verfügbar, sende Anfrage an c. Gelingt keiner der Kommunikationsversuche, starte den Algorithmus wieder bei a.

Dadurch, dass zu keinem Zeitpunkt Anfragen an zwei Ressourcen gleichzeitig geschickt werden, bleibt die Exklusivität der Summationspfade gewahrt, da die Nachrichten an die Ressourcen dort auch Zustandswechsel auslösen könnten.

Hat die virtuelle Maschine einen Kommunikationsversuch gestartet und wiederholt sie die Anfrage periodisch, sollte der Prozess die Möglichkeit besitzen, diese Kommunikation abzubrechen. Dazu wird ein Zeitgeberprozess T definiert, der nach dem Eingang einer Nachricht eine eingestellte Zeit wartet, bevor er dem Sender antwortet.

$$SYS \stackrel{def}{=} \quad \mathbf{v}timer\ (T \mid P) \tag{4.3}$$

$$T(timer) \stackrel{def}{=} \quad !timer(t).\tau.send(t) \tag{4.4}$$

$$P(timer) \stackrel{def}{=} \quad \mathbf{v}(a,t)\ \overline{timer}\langle t\rangle(\overline{a}.Q + t) \tag{4.5}$$

Hier sendet der Prozess vor dem Start der eigentlichen Kommunikation den Namen t an den Zeitgeberprozess. Blockiert die Kommunikation über a länger, als die Wartezeit im Prozess T (Das Warten wird hier durch das τ abstrahiert. Siehe dazu den nächsten Abschnitt), so ist das Lesen vom Namen t vor dem Senden über a erfolgreich. Der Kommunikationsversuch wird somit abgebrochen, da dieser Pfad der Summation verworfen wird.

Es wurde gezeigt, das π-Prozesse als Namen auch URIs verwenden können und so mit Hilfe der virtuellen Maschine und eines HTTP-Servers über das Internet untereinander und mit anderen Clients und Servern kommunizieren können. Die π-Prozesse können beide Rollen der Client-Server-Kommunikation annehmen. Es wurde außerdem gezeigt, wie das verbindungsorientierte und nichtblockierende Kommunikationsverhalten mit dem π-Kalkül abgebildet werden kann.

Im folgenden Abschnitt wird auf die Einbindung von Funktionen eingegangen, die in anderen Sprachen als dem π-Kalkül formuliert worden sind.

4.3 Einbinden von Nicht-π-Funktionen

In der Theorie des π-Kalküls wird fast ausschließlich über die Prozesse an sich diskutiert. Funktionale Aspekte werden als τ ausgeklammert. Soll nun eine Implementierung realisiert werden, die π-Prozesse als Beschreibung von Anwendungsverhalten nutzt, muss es auch die Möglichkeit geben, nicht in π spezifiziertes Verhalten in die Prozessausführung zu integrieren. Obwohl es durchaus als möglich er-

scheint, alle Teile einer Anwendung in π zu beschreiben [36, 48], ist es wohl sinnvoller zum Beispiel mathematische Berechnungen oder die Datenhaltung in dafür besser geeignete Sprachen zu implementieren. Mathematische Operationen lassen sich so direkt auf einem Prozessor mit hoher Geschwindigkeit ausführen und für die persistente Verwaltung von Informationen können ausgereifte Datenbank-Management-Systeme mit der Standard-Anfrage-Sprache SQL[28] genutzt werden.

Eine Implementierung benötigt eine Schnittstelle, um Funktionen in π-Prozesse einbinden zu können. Dazu kann das stille Präfix τ verwendet werden. Die Reduktionsregel zu τ besagt, dass $\tau.P$ unter allen Umständen zu P reduziert werden kann. Die Schnittstelle könnte nun so aussehen, dass jeweils mit einem τ eine Funktion aufgerufen werden kann. Um in der Prozessbeschreibung festlegen zu können, welche Funktion aufgerufen werden soll, muss das τ mit dem Namen der Funktion annotiert werden. $\tau[name].P$ ruft die Funktion mit dem Namen $name$ auf und verhält sich nach dem Fertigstellen des Funktionsaufrufes wie P. Sollen Parameter an die Funktion übergeben werden bzw. werden Rückgabewerte erwartet, sieht die Notation wie folgt aus:

$$\tau[name]\langle param1, param2\rangle(ergebnis1, ergebnis2) \tag{4.6}$$

Alle Parameter und Ergebnisse sind π-Namen. Die Parameter werden an die Funktion gesendet und die Ergebnisse werden gelesen. Wird die Funktion als Prozess angesehen, die ihre Parameter vom Kanal $name$ liest, wäre eine gültige Kodierung in den polyadischen π-Kalkül:

$$\mathbf{v}ret\ .\overline{name}\langle param1, param2, ret\rangle.ret(ergebnis1, ergebnis2) \tag{4.7}$$

Da die Ergebnisse der Funktion gelesen werden, müssen die Namen $ergebnis1$ und $ergebnis2$ durch die tatsächlich gelesenen Namen ersetzt werden.

Eine einfache Beispielfunktion ist der Timer aus dem vorangehenden Abschnitt. Die Wartefunktion wurde dort durch ein einfaches τ abstrakiert. Nun kann die Funktion $wait$, die eine vorgegebene Zeit wartet, mit $\tau[wait]$ eingebunden werden. Da auch Zahlen im π-Kalkül ausgedrückt werden können [48], kann die Funktion auch

parametrisiert werden. Mit $\tau[wait]\langle 30\rangle$ etwa könnte die Wartezeit auf 30 Sekunden gesetzt werden.

4.4 Sicheres und idempotentes Verhalten

Die einheitliche Schnittselle von REST dient vor allem dazu, dass alle an der Kommunikation Beteiligten, d.h. sowohl Client und Server als auch alle Stationen dazwischen, die Semantik der Nachricht erkennen können. Unterteilt wird in Nachrichten, die ein sicheres, ein idempotentes oder ein nicht-idempotentes Verhalten auf dem Server auslösen [20]. Dazu muss die Einhaltung der Semantik auf Serverseite sichergestellt sein. Ruft der Client beispielsweise ein sicheres Verhalten auf und löst das Verhalten auf dem Server Seiteneffekte auf, kann der Client dafür nicht verantwortlich gemacht werden.

Durch die formale Semantik des π-Kalküls wäre es interessant, π-Prozesse automatisiert auf ihr Verhalten hin zu untersuchen. Es wird ein Verfahren gesucht, mit dem gezeigt werden kann, ob das Verhalten eines π-Prozesses sicher oder idempotent ist.

Ein Serverprozess S zeigt bei der Kommunikation mit einem Clientprozess C ein sicheres Verhalten, wenn der Prozess vor und nach der Bearbeitung der Anfrage identisch, das heißt, strukturell kongruent ist. Für den Prozess muss gelten:

$$S \mid C \longrightarrow S' \mid C' \Rightarrow S' \equiv S \tag{4.8}$$

Ferner muss sichergestellt sein, dass in aufgerufenen Funktionen, die, wie oben beschrieben, über τ eingebunden werden können, keine relevanten Daten im Datenhaltungssystem verändert werden. Ein möglicher Ansatz dazu wäre, dass die Funktionen als sicher oder unsicher markiert werden, sodass automatisiert geprüft werden kann, ob ein Prozess nur sichere Funktionen aufruft, die keine Daten ändern.

Bei idempotenten Verhalten ist bei der ersten Nachricht eine Änderung von Daten und des Prozesses selbst gestattet, jedoch dürfen bei allen Wiederholungen der gleichen Nachricht keine Änderungen mehr auftreten.

$$S \mid C \longrightarrow S' \mid C' \Rightarrow S' \mid C \longrightarrow S'' \mid C' \wedge S'' \equiv S' \tag{4.9}$$

Kommuniziert S mit C, entwickeln sich daraus S' und C'. S' darf nun keine unsicheren Funktionsaufrufe mehr beinhalten und bei einer erneuten Kommunikation muss sich C wieder zu C' entwickeln und S' darf sich strukturell nicht weiter verändern.

Diese Regeln gelten für die Kommunikation zwischen einem Serverprozess und einem Clientprozess. Typischerweise beginnt der Serverprozess mit einer Replikation und liest dann eine eingehende Anfrage über einen URI, während der Clientprozess die Nachricht an diesen URI sendet.

$$S(uri) \stackrel{def}{=} \quad !uri(o, req).T(o, req) \tag{4.10}$$

$$C(uri) \stackrel{def}{=} \quad \mathbf{v}o\, \overline{uri}\langle o, req\rangle.o(resp).C' \tag{4.11}$$

In diesem Fall muss gezeigt werden, dass T sich zu $\mathbf{0}$ entwickelt und währenddessen die Antwort über Kanal o an den Client sendet.

$$S \mid C \longrightarrow S \mid T \mid o(resp).C' \Rightarrow T \mid o(resp).C' \longrightarrow 0 \mid C' \tag{4.12}$$

Es genügt zu zeigen, dass T den Prozess Q schwach offen D-simuliert $Q \approx_O^D T$.

$$Q(o, req) \stackrel{def}{=} \overline{o}\langle resp\rangle.\mathbf{0} \tag{4.13}$$

Diese Überlegungen stellen einen Anfang dar, um Prozesse auf sicheres und idempotentes Verhalten hin zu untersuchen. Es bleibt noch zu zeigen, wie das Verhalten eines Serverprozesses formal verifiziert werden kann, wenn viele Clients gleichzeitig mit dem Prozess kommunizieren. Dies würde jedoch den Rahmen dieser Arbeit sprengen und sollte daher Bestandteil von weiteren Forschungsarbeiten sein.

4.5 Zusammenfassung

In diesem Kapitel wurde erörtert, wie die grundlegenden Aspekte von REST im π-Kalkül beschrieben werden können. Dazu zählt die Verwendung von URIs als π-Namen und die bidirektionale Kommunikation über HTTP mit Abbildung der blockierenden Semantik von π sowohl in der Client- als auch in der Serverrolle. Ferner wurde dargelegt, wie Funktionen in π-Prozesse eingebunden werden können, was insbesondere für den Einsatz des π als Implementierung von Ressourcenverhalten wichtig ist. Zuletzt wurde ein formaler Exkurs unternommen, in dem ein Ansatz zur Untersuchung von Prozessen auf sicheres oder idempotentes Verhalten vorgestellt wurde.

All diese theoretischen Überlegungen bescheinigen die Verwendbarkeit des π-Kalküls in der Ressourcenorientierung. Um die anfangs gestellte Frage, ob sich das Verhalten von Ressourcen durch den π-Kalkül implementieren lässt, endgültig zu beantworten, soll die vorgestellte Theorie ihre Brauchbarkeit unter Beweis stellen. Dazu wird im nächsten Kapitel eine Implementierung der beschriebenen Ansätze entwickelt.

Kapitel 5

Eine virtuelle Maschine für den π-Kalkül

Zur Überprüfung der Hauptthese dieser Arbeit wird in diesem Kapitel eine virtuelle Maschine vorgestellt, die Beschreibungen von Ressourcenverhalten durch den π-Kalkül direkt interpretieren kann. Zuerst werden die Anforderungen an und die Designziele für eine solche virtuelle Maschine herausgearbeitet und ein Ansatz entworfen, wie diese technisch umgesetzt werden können. Anschließend werden die nicht-trivialen Implementierungsdetails veranschaulicht. Doch vorher wird mit einer kleinen allgemeinen Einführung zu virtuellen Maschinen begonnen.

5.1 Virtuelle Maschinen

Es existieren zwei grundlegende Verfahren, um Programmcode zur Ausführung zu bringen: kompilieren und interpretieren [16]. Bei der Kompilierung wird der Code in ein architekturspezifisches Format umgewandelt und kann danach direkt von einer (Hardware-)Maschine ausgeführt werden. Normalerweise sind die dazu benötigten Compiler direkt auf eine Architektur zugeschnitten, zum Beispiel auf die x86-Architektur. Ein Interpreter hingegen führt den Programmcode direkt aus, was zum Vorteil hat, dass ein Programm ohne Anpassungen auf jeder Plattform lauffähig ist, auf der ein entsprechender Interpreter existiert. Der Code wird dadurch portabel.

Virtuelle Maschinen kombinieren beide Ansätze. Der Programmcode wird zuerst durch einen Compiler in ein internes Format mit einem festgelegten Instruktionssatz übersetzt. Der Kern der virtuellen Maschine implementiert jede Instruktion dieses Formates und kann so das übersetzte Programm ausführen, was auch als Interpretation oder Emulation angesehen werden kann. Bei der Übersetzung des Programmcodes werden meistens Sprachkonstrukte von einer hohen Abstraktionsebene auf einfachere Elemente abgebildet und dadurch der Instruktionssatz verkleinert und die Implementierung der Maschine vereinfacht. Zum Beispiel können verschiedene Schleifentypen im Programmcode durch bedingte Sprünge im Kompilat realisiert werden.

Wird die Vision aus der Einführung betrachtet, so können Geschäftsprozesse durch eine virtuelle Maschine ausgeführt werden, bei der der π-Kalkül als internes Format gewählt wird. Puhlmann beschreibt in [48] eine Abbildungsvorschrift von Workflowpattern auf den π-Kalkül und hat in diesem Zusammenhang einen prototypischen Compiler implementiert. Da der hier verwendete π-Kalkül aus nur neun Instruktionen besteht, deren Semantik durch die Theorie festgelegt ist, sollte eine virtuelle Maschine diesen Instruktionssatz relativ einfach implementieren können.

Ziel für die hier vorgestellte virtuelle Maschine ist es, einen lauffähigen Prototypen zu realisieren, der alle π-Elemente unterstützt. Dadurch sollten sich alle Prozesse, die mit dem π-Kalkül beschrieben werden können, ausführen lassen. Mögliche Erweiterungen des Instruktionssatzes zur flexibleren Gestaltung von π-Prozessen oder zur Steigerung der Interpretationsgeschwindigkeit werden hier außen vorgelassen, da das Hauptaugenmerk auf der Evaluierung der Machbarkeit einer solchen Maschine ruht.

5.2 Anforderungen an die virtuellen Maschine

Die Aufgabe ist es, eine virtuelle Maschine zur Ausführung von π-Prozessen als Implementierungen von Ressourcenverhalten zu realisieren. Daraus ergeben sich folgende Anforderungen.

- **Semantik** Die Semantik des π-Kalküls muss eingehalten werden, da sonst Erkenntnisse aus der formalen Verifikation von Prozessen nicht auf die Aus-

führung dieser Prozesse übertragen werden können.

- **Kommunikation** Ressourcen im Internet sollen über HTTP angesprochen werden können und auch andere Ressourcen selber über HTTP ansprechen können.

- **Parallelität** Gerade im Serverbereich ist es üblich, dass die Hardware mehrere Threads gleichzeitig verarbeiten kann. Um diese Möglichkeiten zu nutzen, soll die virtuelle Maschine parallele π-Prozesse auch nebenläufig ausführen (Multithreading).

- **Skalierbarkeit** Ein Hauptziel von REST ist eine hohe Skalierbarkeit. Dem soll die virtuelle Maschine Rechnung tragen. Bei der Wahl von Algorithmen ist auf eine geringe Laufzeitkomplexität zu achten.

5.3 Entwurfsentscheidungen

Eine virtuelle Maschine für diese Anforderungen kann sicherlich auf verschiedene Weisen realisiert werden. In diesem Abschnitt werden die Entwurfsentscheidungen und ihre Gründe erläutert, die für die in dieser Arbeit vorgestellte Umsetzung getroffen wurden.

5.3.1 Darstellung der Prozesse

Um π-Prozesse in der virtuellen Maschine zu bearbeiten, müssen sie in einem geeigneten Format vorliegen. Bog zeigt in ihrer Masterarbeit [11], wie π-Prozesse als Baumstruktur dargestellt werden können. Jedes der π-Elemente ! $\mid$ + $\mathbf{v}$ $\overline{x}\langle\rangle$ $y()$ $A(\tilde{z})$ wird jeweils als ein Knoten dargestellt. Die möglichen nachfolgenden Prozessschritte werden als Kinder dieser Knoten realisiert. Wie in Abbildung 5.1 zu sehen, wird eine Sequenz wie $\mathbf{v}a\,\overline{x}\langle a\rangle.a(y)$ durch die Verkettung von Knoten ausgedrückt (a). Bei einer Summation wie $a(x)+b(y)$ wird einer der Kindknoten ausgeführt (b), wohingegen bei parallelen Prozessen wie $a(x) \mid b(y)$ alle Kindknoten nebenläufig ausgeführt werden (c).

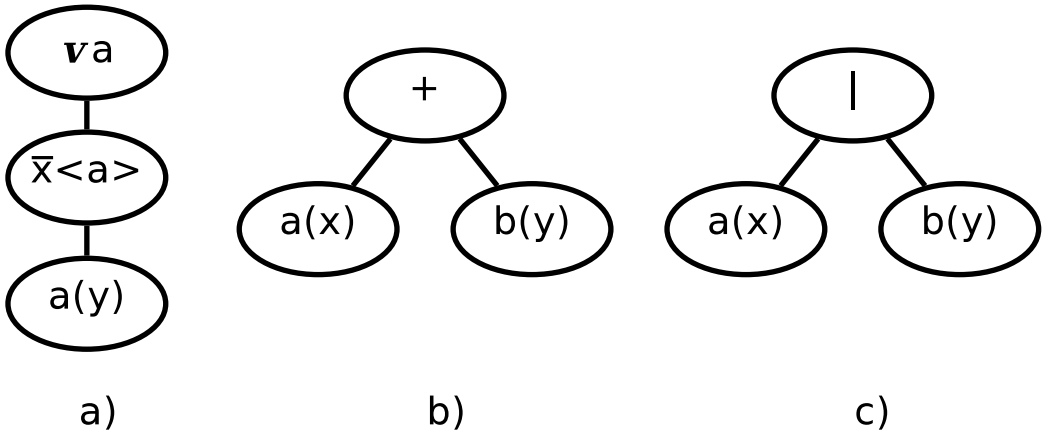

Abbildung 5.1: Darstellung von Sequenzen, Summationen und Parallelität

Bei der Abarbeitung von π-Prozessen ‚entwickeln' sich die Prozesse stetig weiter, das heißt, dass abgearbeitete Prozessschritte und mit ihnen eventuell ganze Teilbäume verschwinden, so etwa bei der Auswahl einer Verzweigung einer Summation oder durch den Aufruf eines anderen Prozesses die Prozessbeschreibung erweitert wird. Außerdem müssen Namensersetzungen vorgenommen werden, um die Semantik der gebundenen Namen umzusetzen.

Um die Anforderung nach Skalierbarkeit zu befriedigen, muss eine Lösung gefunden werden, bei der die für den π-Kalkül erforderlichen Transformationen der Prozesse und die Namensersetzungen mit einer geringen Laufzeitkomplexität erfolgen. Dazu soll folgende Annahme getroffen werden: Die virtuelle Maschine soll als Implementierung von Diensten im Internet verwendet werden, zum Beispiel für den in Kapitel 3 beschriebenen Onlineshop. Typischerweise rufen dabei viele Clients dieselben Ressourcen auf und starten dieselben Prozesse, etwa den Prozess der Bestellabwicklung. Für jeden Client wird ausgehend von derselben Prozessbeschreibung ein eigener Prozess gestartet und abgearbeitet. Da der Begriff Prozess mehrdeutig aufgefasst werden kann, soll die Bedeutung für die Beschreibung der virtuellen Maschine hier festgelegt werden.

Definition 16 Eine *Prozessbeschreibung* ist ein Modell, das die Abfolge mehrerer Aktivitäten beschreibt. △

Definition 17 Ein *Prozess* ist eine konkrete Abarbeitung einer Prozessbeschrei-

bung, die laufzeitspezifische Daten enthält. △

Im beschriebenen Szenario liegen dem System mehrere Prozessbeschreibungen, formuliert durch den π-Kalkül, vor, so zum Beispiel die Beschreibung der Bestellabwicklung. Für jeden Kunden oder jede Kundin wird nach dem Abschicken der Bestellung im Onlineshop jeweils ein Prozess gestartet, der die Prozessbeschreibung der Bestellabwicklung abarbeitet. Die Prozesse laufen normalerweise voneinander unabhängig, so kann jede Bestellung unterschiedliche Waren beinhalten und durch Verzweigungen in der Prozessbeschreibung kann der Kontrollfluss zwischen den einzelnen Prozessen variieren. Der Zusammenhang zwischen Prozessbeschreibung und Prozess ist analog zu dem Zusammenhang zwischen Klasse und Objekt in der Objektorientierung.

Es wird bei der Nutzung der virtuellen Maschine für die Realisierungen von Diensten häufig vorkommen, dass viele Prozesse zu einer Prozessbeschreibung gestartet werden. Da sich im π-Kalkül bei der Abarbeitung eines Prozesses dessen Beschreibung verändert, könnte ein erster Lösungansatz sein, die Beschreibung für jeden neu gestarteten Prozess zu kopieren, sodass der Prozess exklusiv auf seiner Beschreibung arbeiten kann. Er könnte π-Namen, die in der Beschreibung vorkommen, nach den Regeln der Namensersetzung für gebundene Namen ersetzen und abgearbeitete Aktivitäten oder Verzweigungen entfernen, ohne andere Prozesse zu beeinflussen.

Doch auf der Suche nach einer möglichst skalierbaren Lösung erscheint das häufige Kopieren und Ändern von Baumstrukturen ungeeignet. Der in dieser Arbeit verwendete Ansatz geht davon aus, dass die Beschreibung eines Prozesses nicht kopiert wird, sondern nur einmal im System existiert und alle Prozesse auf diese Beschreibung verweisen. Dabei zeigt jeder Prozess auf den gerade von ihm ausgeführten Schritt in der Prozessbeschreibung, das heißt auf einen Knoten in der Baumstruktur. Hat der Prozess einen Schritt abgearbeitet, wird einfach nur die Referenz auf den nächsten Knoten der Prozessbeschreibung ‚weitergeschoben'. Dabei ‚sieht' ein Prozess von der Prozessbeschreibung nur den aktuellen Knoten und alle Unterknoten. Wird etwa bei einer Summation eine Verzweigung ausgewählt, wird die Referenz auf den ersten Knoten des entsprechenden Teilbaums gesetzt. Die übrigen Verzweigungen sind für diesen Prozess nicht mehr sichtbar, wurden

also eliminiert, ohne die eigentliche Baumstruktur verändern zu müssen. Prozessspezifische Daten können bei diesem Ansatz nicht in die Prozessbeschreibung eingefügt werden, da der Prozess nicht mehr exklusiv über sie verfügt. Für die Namensersetzung kann für jeden Prozess eine Tabelle genutzt werden, in der vermerkt wird, welcher π-Name durch welchen π-Namen ersetzt wird. Wird zum Beispiel in einem Prozess der Name a aus der Prozessbeschreibung durch den Namen b ersetzt, so wird nicht mehr die Prozessbeschreibung traversiert und alle Vorkommen des Namens angepasst, sondern die Ersetzung wird in die Ersetzungstabelle eingetragen. Als Konsequenz muss bei der Nutzung eines Namens aus der Prozessbeschreibung jedes Mal in der Tabelle nachgeschaut werden, ob es für diesen Namen eine Ersetzung gibt.

Der eben beschriebene Ansatz hat gegenüber dem ersten Lösungsansatz verschiedene Vorteile. Zum Einen wird jede Prozessbeschreibung nur ein einziges Mal im System vorgehalten, was Speicher einspart. Ein Prozess besteht nur noch aus einer Ersetzungstabelle und einem Verweis auf einen Schritt in der Prozessbeschreibung. Dadurch können viele Prozesse parallel auf einem Server ausgeführt werden, ohne viel Speicher zu beanspruchen. Desweiteren muss bei keinem Prozessschritt ein Baum komplett traversiert werden, um zum Beispiel Namen zu ersetzen. Wird die Ersetzungstabelle als Hash-Tabelle ausgeführt, besitzen sowohl das Weiterrücken der Referenz auf die Prozessbeschreibung, das Aufnehmen einer Ersetzung sowie das Nachschlagen einer Ersetzung in der Tabelle eine konstante Laufzeitkomplexität.

Jedoch ist zu berücksichtigen, dass die erwarteten Eigenschaften auf der Annahme beruhen, dass viele Prozesse dieselbe Prozessbeschreibung nutzen. Existieren hingegen im System viele Prozesse, die jeweils unterschiedliche Prozessbeschreibungen abarbeiten und benötigen diese Beschreibungen viel Speicher, könnte es sinnvoller sein, die Baumstrukturen zu manipulieren und deren Größe durch die Entwicklung der Prozesse stetig zu verringern. Außerdem können Probleme bei der Aktualisierung von Prozessbeschreibungen entstehen, da zu jedem Zeitpunkt viele Prozesse auf die Beschreibung referenzieren können. Um dieses Problem zu umgehen, könnte eine Versionierung der π-Prozessbeschreibungen eingeführt werden. Ein Prozess referenziert dann auf eine Beschreibung in einer bestimmten Version. Wird die Beschreibung geändert, wird eine neue Version er-

stellt, die alle neu instanziierten Prozesse nutzen, während die alten Prozesse noch auf die alte Version zugreifen können.

5.3.2 Realisierung von Parallelität

Der π-Kalkül ist auf konkurrierende, nebenläufige Prozesse ausgerichtet. Mit dem hier vorgestellten Entwurf soll eine hohe Nebenläufigkeit der Prozessabarbeitung erreicht werden. Mögliche Engpässe, die das gesamte System ausbremsen könnten, sollen vermieden werden.

Da die Anzahl der möglichen nebenläufigen Prozesse in einem π-System nicht festgelegt ist, jedoch die Anzahl der möglichen Threads bzw. Systemprozesse eines Betriebsystem begrenzt ist, erfolgt eine m:n-Zuordnung zwischen π-Prozessen und Betriebsystemthreads. Die sinnvolle Zahl der genutzten Threads ist meistens noch deutlich geringer, da sie von der Anzahl der von der Hardware gleichzeitig ausführbaren Threads abhängt. Da diese Eigenschaft systemabhängig sind, sollte die Zahl der genutzten Threads in der virtuellen Maschine konfigurierbar sein.

Prozesse sollen in der virtuellen Maschine die kleinste Einheit der Nebenläufigkeit sein. Tritt in einem Prozess ein parallele Aufteilung auf, so wird die Datenstruktur des Prozesses (inklusive der Ersetzungstabelle) dupliziert und beide Prozesse werden dann unabhängig voneinander weiter abgearbeitet.

Die Warteschlange

Für die Aufteilung der Prozesse auf Systemthreads kann eine Warteschlange genutzt werden. Alle π-Prozesse der virtuellen Maschine, die gerade nicht bearbeitet werden, warten in dieser Schlange. Daneben gibt es eine Anzahl von Arbeiterthreads, die jeweils in einem Betriebsystemthread oder -prozess laufen. Jeder dieser Arbeiterthreads entnimmt der Warteschlange einen Prozess, bearbeitet ihn und fügt ihn anschließend wieder am Ende der Warteschlange an, um den nächsten Prozess zu bearbeiten. Bei der Implementierung der Warteschlange muss darauf geachtet werden, dass sich die Arbeiterthreads beim Entnehmen von Prozessen aus und beim Anhängen an die Warteschlange nicht gegenseitig unnötig lange blockieren, um die parallele Ausführung der Prozesse nicht zu behindern.

5.3.3 Kommunikation

Die Umsetzung der Kommunikation des π-Kalküls ist durch ihr blockierendes Verhalten nicht ganz trivial. Ein Prozess, der mit einem anderen Prozess kommunizieren möchte, muss warten, bis der Kommunikationspartner bereit ist. Die Realisierung kann durch eine erweiterte Warteschlange geschehen, die die Information enthält, ob ein wartender Prozess lesen oder schreiben will und welcher Kanal für die Kommunikation genutzt werden soll. Tritt in einem Prozess P eine Kommunikation auf, zum Beispiel $\overline{x}\langle a\rangle$, so muss der Arbeiterthread zuerst in der Kommunikationswarteschlange nach einem Prozess suchen, der vom Kanal x lesen möchte. Wird er fündig, wird der wartende Prozess aus der Warteschlange entnommen und der Arbeiterthread führt die Kommunikation zwischen beiden Prozessen aus. Anschließend wird der entnommene Prozess in die normale Warteschlange eingetragen, sodass ein freier Arbeiterthread sich um dessen Abarbeitung kümmern kann. Der aktuelle Arbeiterthread wird den Prozess P weiterführen. Steht jedoch in der Kommunikationswarteschlange kein zur gewünschten Kommunikation passender Prozess bereit, so muss der Prozess P selber in die Kommunikationswarteschlange eingefügt werden und auf einen geeigneten Kommunikationspartner warten.

Auch bei der Implementierung der Kommunikationswarteschlange muss darauf geachtet werden, dass sich mehrere Arbeiterthreads beim Zugriff auf die Warteschlange so wenig wie möglich blockieren und dennoch keine Inkonsistenzen auftreten.

5.3.4 π-Namen

Wie im vorangehenden Kapitel beschrieben, werden als Namen zur Kommunikation über das Internet URIs verwendet. Um die Eindeutigkeit der URIs zu gewährleisten, soll der Server die URIs verwalten. Die Implementierung des Restriktionsoperators fragt vor der Verwendung eines neuen URI, ob dieser schon existiert. Ist dies der Fall, gibt der Server einen neuen URI zurück, der bisher noch nicht verwendet wird. Der Server achtet auch darauf, dass nur Namen erzeugt werden, die seinen Hostnamen als Domäne enthalten, also die durch die Infrastruktur auf ihn abgebildet werden.

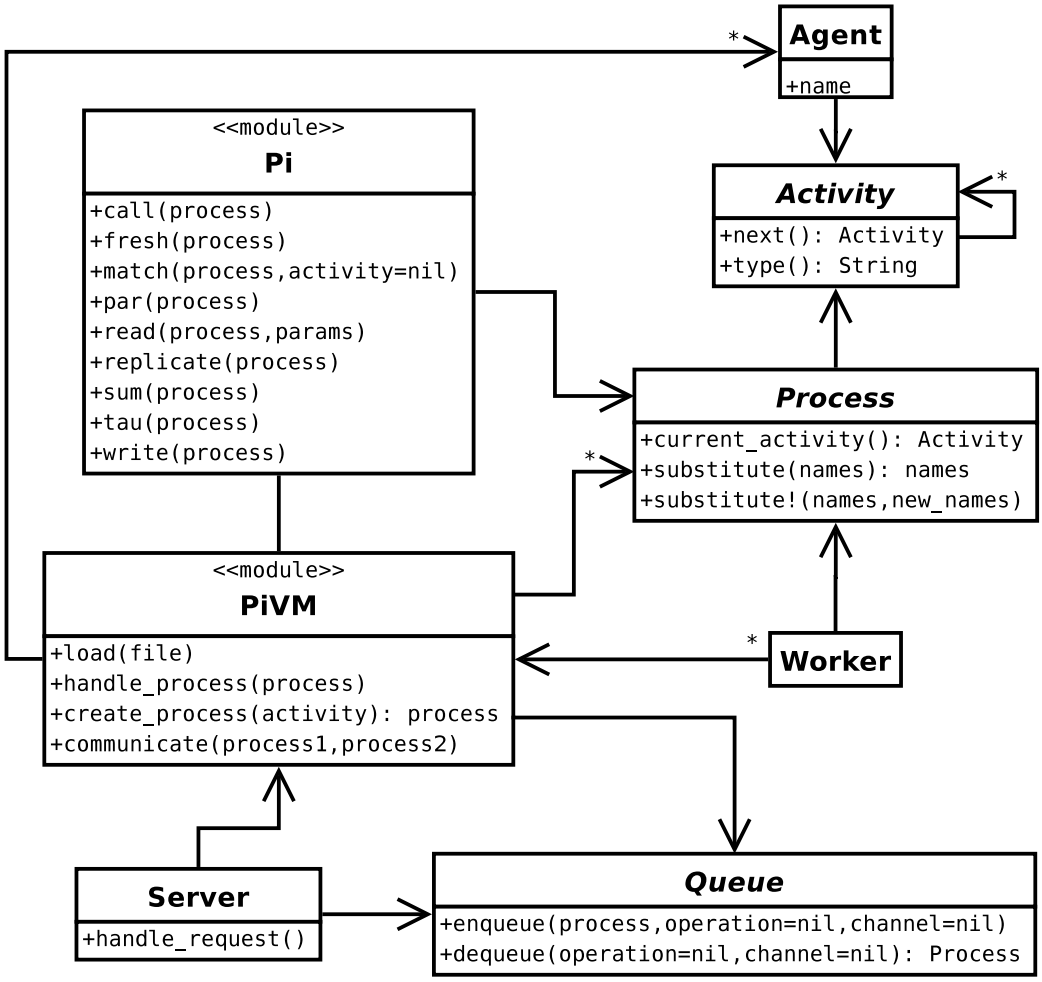

Abbildung 5.2: UML-Klassendiagramm der virtuellen Maschine

5.4 Implementierung

In diesem Abschnitt sollen die nicht-trivialen Implementierungsdetails der virtuellen Maschine veranschaulicht werden. Als Programmiersprache wurde Ruby [50] verwendet, die eine schnelle Implementierung des Prototypen gestattete. Das UML-Klassendiagramm mit den wichtigsten Klassen ist in Abbildung 5.2 dargestellt. Die Prozessbeschreibungen liegen in Form von Agenten vor, wobei jeder Agent auf genau eine Startaktivität verweist. Jede Aktivität entspricht einem π-Syntaxelement und verweist auf die Folgeaktivitäten im π-Prozess. Parallele Aufteilungen und Summationen können mehrere Folgeaktivitäten besitzen, während Prozessaufrufe

und Terminierung auf keine weiteren Aktivitäten verweisen. Alle anderen Aktivitäten besitzen genau eine Folgeaktivität. Durch diese Aneinanderkettung von Aktivitäten können Prozessbeschreibungen als Baum dargestellt werden. Jedes Exemplar der Klasse *Process* stellt die Ausführung einer Prozessbeschreibung dar und referenziert die auszuführende Aktivität. Über die Methode *substitute!()* können neue Namensersetzungen in die Ersetzungstabelle des Prozesses aufgenommen und mit *substitute()* wieder abgefragt werden. Die Hauptklasse der virtuellen Maschine ist das Modul *PiVM*. Es verwaltet alle Agenten und Prozesse, kann neue Prozessbeschreibungen laden und über *handle_process()* einen Prozessschritt abarbeiten. Die Hauptaufgabe dieser Methode besteht in der Behandlung der synchronen Kommunikation zweier Prozesse, für die immer eine Lese- und eine Schreibaktivität ausgeführt werden muss, und der Replikation. Die eigentliche Behandlung der einzelnen π-Aktivitäten erfolgt im Modul *Pi*, in dem für jedes π-Element eine statische Methode implementiert ist.

Die Klasse *Worker* stellt einen Arbeiterprozess dar. Jeder Worker läuft in einem eigenen Betriebssystemprozess und führt immer genau einen *Process* aus. Kann ein Prozess nicht weiter ausgeführt werden, da er zum Beispiel auf einen Kommunikationspartner warten muss, wird er in die Warteschlange (*Queue*) eingetragen und der Worker entnimmt sich daraus den nächsten Prozess zur Bearbeitung. Im Folgenden wird die Umsetzung zu einigen nicht-trivialen Aspekten geschildert.

Restriktion

Die Restriktion (im Programmcode als *fresh* bezeichnet) muss nach Abschnitt 4.2.1 für die Eindeutigkeit von erzeugten URIs sorgen. Die hier umgesetzte einfachste Variante hängt an jeden zu erzeugenden URI eine eindeutige, mehrstellige Zahl, die bei jedem Aufruf des Restriktionsoperator um eins inkrementiert wird. Für spätere Versionen ist es anzustreben, für Menschen lesbarere URIs zu erzeugen, die ohne angehängte Zahl auskommen. Die Eindeutigkeit könnte durch die Pflege einer Liste alle bereits erzeugter URIs sichergestellt werden.

Replikation

Durch die verwendete π-Syntax aus Tabelle 2.3 kann eine Replikation nur direkt vor einer Kommunikation auftreten. Bei der Bearbeitung muss dabei zwischen zwei Situationen unterschieden werden. Wird ein Prozess abgearbeitet und trifft die virtuelle Maschine auf eine Replikationsaktivität, liest die virtuelle Maschine die folgende Kommunikationsaktivität und trägt den Prozess als einen auf Kommunikation wartenden Prozess in die Warteschlange ein und markiert ihn zusätzlich als Replikation. Anschließend schaut die virtuelle Maschine in der Warteschlange nach, ob bereits Prozesse auf diese Kommunikation gewartet haben, entnimmt diese Prozesse der Warteschlange, erzeugt eine Kopie des zu replizierenden Prozesses für jede wartende Kommunikation, führt die entsprechenden Kommunikationen aus und hängt dann alle Prozesse wieder in die Warteschlange ein, sodass sie von den Workern weitergeführt werden können.

Die zweite Situation entsteht nun, wenn später zu einem Prozess in der Warteschlange ein Kommunikationspartner gefunden wird, der als Replikation markiert worden ist. Dann verbleibt dieser in der Warteschlange und die virtuelle Maschine erzeugt stattdessen eine Kopie von diesem Prozess, bei der dann die Kommunikation ausgeführt wird. Die Kopie wird nun zur weiteren Bearbeitung in die Warteschlange gehängt.

Parallele Aufteilung

Bei einer parallelen Aufspaltung eines Prozesses wird bei n Folgeaktivitäten $n-1$ Kopien des Prozesses angelegt und jeder der nun n Prozesse führt eine der Folgeaktivitäten aus. Beim Kopieren wird die Ersetzungstabelle des Prozesses mit dupliziert und die Referenz auf die jeweilige Folgeaktivität in der Prozessbeschreibung gesetzt. Der erste Prozess wird dann durch den aktuellen Workerthread weiter ausgeführt, alle weiteren werden in die Warteschlange gehängt.

Summation

Dem gegenüber ist die Summation deutlich komplizierter umzusetzen, da bei einer Summation eine Auswahl zwischen einer der Folgeaktivitäten getroffen wer-

den muss. Der verwendete Algorithmus funktioniert wie folgt: Die virtuelle Maschine beginnt mit der ersten Folgeaktivität. Handelt es sich dabei um einen Vergleichsoperator (*Match*), wird der Vergleich der angegebenen π-Namen durchgeführt. Sind sie ungleich, wird dieser Pfad verworfen und die virtuelle Maschine untersucht die nächste Folgeaktivität der Summation. Ist der Vergleich jedoch positiv, wird dessen nächste Folgeaktivität untersucht, da mehrere Vergleichsaktivitäten nacheinander folgen können. Wurden alle vorhandenen Vergleiche als positiv entschieden, wird die folgende Aktivität untersucht. Ist es ein τ, dass zu jeder Zeit ausgeführt werden kann, wählt die Summation diesen Pfad und verwirft alle anderen. Ist es eine Kommunikation, wird diese in eine Liste aufgenommen und der nächste Pfad der Summation wird untersucht.

Waren unter allen Pfaden keine τ, sondern Kommunikationen, so entsteht durch den Algorithmus eine Liste dieser Kommunikationsaktivitäten. Nun wird geprüft, ob sich in der Warteschlange ein passender Kommunikationspartner zu einer der Kommunikationen befindet. Wenn ja, wird dieser entnommen, die entsprechende Kommunikation ausgeführt und alle weiteren Alternativen werden verworfen. Wird kein Kommunikationspartner gefunden, werden alle alternativen Kommunikationen in die Warteschlange eingetragen. Durch ihre gemeinsame Prozess-ID sind die einzelnen Kommunikationen als Bestandteile einer Summation zu identifizieren.

Führt ein anderer Prozess eine Kommunikation aus und entnimmt dazu eine wartende Kommunikation aus der Warteschlange, so werden gleichzeitig alle anderen wartenden Kommunikationen mit derselben Prozess-ID (denn das sind dann die anderen Alternativen einer Summation) gelöscht.

τ

Das erweiterte τ ermöglicht nach den Konzepten aus Kapitel 4 das Aufrufen von Funktionen mit Parametern. Für die Umsetzung existiert eine Klasse, in der Funktionen für π-Prozesse definiert werden können. Die virtuelle Maschine ruft für ein τ mit einer angegebenen Funktion die Funktion mit demselben Namen aus dieser speziellen Klasse auf, übergibt die Parameter an diese Funktion und führt die Ergebnisse anschließend als Ersetzung für die Platzhalter in die Ersetzungstabelle des

π-Prozesses ein.

Multithreading

Threads in Ruby sind als *grüne Threads* implementiert. Das heißt, dass der Ruby-Interpreter die komplette Threadbehandlung selber implementiert und in diesem Punkt nicht mit dem Betriebssystem zusammenarbeitet. Das hat den Vorteil, dass die Threads in Ruby einfach zu benutzen und portabel sind, jedoch lassen sie sich nicht auf mehrere Prozessoren oder Prozessorkerne verteilen. Es wird derzeit diskutiert, wie das Threading-Modell in Ruby in der Zukunft aussehen wird [53]. Für die virtuelle Maschine wurden daher zur Realisierung der parallelen Abarbeitung von Prozessen Betriebssystemprozesse verwendet, die auf Unix-Plattformen mit den Systemaufruf `fork()` erzeugt werden können. Die Prozesse müssen untereinander nur beim Zugriff auf die Warteschlange synchronisiert werden. Sie wurde als Datenbanktabelle realisiert, wodurch die Sicherstellung der Konsistenz der Daten und die Synchronisation der Prozesse mit geringem Aufwand realisiert werden konnten. Die benutzten Abfragelogiken wurden in eine Datenbankprozedur gelagert, sodass jeder Arbeiterprozess nur diese Prozedur aufrufen muss.

Persistentes DOM

Für die Ablage der Baumstrukturen und Prozesse wird ein persistentes Document Object Model (DOM) verwendet. Das DOM [64] ermöglicht eine einfache Erstellung und Bearbeitung von baumstrukturierten Daten wie XML-Dateien und auch von den oben beschriebenen Prozessbeschreibungen. Über XPATH [65] sind auch komplexere Abfragen auf einem Baum möglich. Um die Persistenz zu realisieren, basierte das System auf einem Datenbankmanagementsystem, das unter anderem nebenläufige Transaktionen, sichere Datenhaltung und Nutzerverwaltung bereitstellt.

Einfache Benchmarks haben jedoch nach der Implementierung gezeigt, dass die intensive Nutzung des DOMs nicht performant genug ist, um die virtuelle Maschine im Produktiveinsatz zufriedenstellend zu benutzen. Das kann unter anderem an dem frühen Entwicklungsstadium des genutzten persistenten DOMs *Xenodot* [39] liegen. Doch für eine Weiterentwicklung der virtuellen Maschine sollte

evaluiert werden, ob andere Ansätze der Datenablage nicht geeigneter in Hinsicht auf die Performanz sind.

Server

Als HTTP-Server wird Mongrel [38] verwendet. Bei Mongrel genügt es, einen Handler zu implementieren, der die HTTP-Anfrage an die virtuelle Maschine weiterreicht und die Antwort eines π-Prozesses entgegennimmt und in das von Mongrel bereitgestellte Response-Objekt einträgt. Danach sendet der Server die Antwort zurück an den Client. Die komplette Verbindungsverwaltung übernimmt der Server.

Für die Bearbeitung einer Anfrage durch einen Prozess, muss dieser auf alle Elemente der Nachricht zugreifen können. Insbesondere sind das Verb der Anfrage (GET, PUT, DELETE oder POST), die URI-Parameter und der eigentliche Inhalt der Nachricht von Bedeutung. Damit π-Prozesse auch auf Informationen aus dem HTTP-Kopf der Anfrage zugreifen können, wandelt der Handler diese Daten in einen π-Prozess um, der nach Übersendung eines Feldnamens den dazugehörigen Wert zurück liefert. Angenommen, der Server erhält folgende HTTP-Anfrage:

```
POST /vm/broker HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

param1=value1&param2=value2
```

Der zugehörige Prozess, der den Zugriff auf den HTTP-Kopf gestattet, sieht dann wie folgt aus:

$$
\begin{aligned}
&!req(name, ret)(\\
&[name = REQUEST\_METHOD]\overline{ret}\langle POST\rangle \\
&+[name = REQUEST\_URI]\overline{ret}\langle /vm/broker\rangle \\
&+[name = CONTENT\_LENGTH]\overline{ret}\langle 27\rangle + ...)
\end{aligned}
\tag{5.1}
$$

Der Prozess besteht aus einer großen Summation, bei der jeder Pfad mit einem Vergleichsoperator versehen ist. Die Vergleichsoperatoren prüfen, ob der an den Prozess übergebene Name einem Schlüssel aus dem HTTP-Kopf entspricht. Ist ein Vergleich positiv, so wird dieser Pfad gewählt und der Wert des Schlüssels aus dem HTTP-Kopf über den Rückgabekanal ret gesendet. Bearbeitet nun ein Prozess eine Anfrage, so kann er die Informationen des HTTP-Kopfes nutzen.

$$A(uri) \stackrel{def}{=} !uri(o, req).\mathbf{v}ret\, \overline{req}\langle REQUEST\_METHOD, ret\rangle.ret(meth). \\ ([meth = POST]\tau.P(o, ret) + [meth = GET]\tau.Q(o, ret)) \tag{5.2}$$

Hier trifft der Prozess eine Entscheidung anhand des Verbs der HTTP-Nachricht. Ist dies ein GET, so wird der Prozess Q gestartet, ist es ein POST, wird der Prozess P angestoßen.

In späteren Versionen sollte die virtuelle Maschine so erweitert werden, dass der Prozess durch eine native Datenstruktur wie einer Hash-Tabelle umgesetzt wird, um eine höhere Geschwindigkeit zu erreichen.

5.5 Evaluierung und Ausblick

Als Beispielanwendung wurde ein einfacher Broker implementiert, der Kunden an ein geeignetes Kreditinstitut vermittelt. Banken können sich dazu beim Broker registrieren und deregistrieren. Der Broker ist als Ressource ausgelegt und die Kommunikation erfolgt ausschließlich über HTTP. Das Verhalten des Brokers ist in π spezifiziert. Die oben eingeführten Konzepte werden dabei genutzt, das heißt, das Verhalten des Prozesses unterscheidet sich nach dem Verb der HTTP-Anfrage und auch nach dem Inhalt einer Nachricht. Der Broker greift über eigene τ-Funktionen auf eine Datenbank zu, um die registrierten Banken zu verwalten. Die Antworten des Brokers sind in der Hypertext Markup Language (HTML) verfasst, so dass sich das Beispiel aus einem Browser heraus nutzen lässt. Eine genaue Beschreibung des Brokers ist im Anhang A zu finden.

Das Beispiel zeigt, dass prozessorientierte Dienste als Ressourcen realisierbar

sind, deren Verhalten mit dem π-Kalkül beschrieben worden sind. Somit wurde die Hauptfrage dieser Arbeit beantwortet. Da die virtuelle Maschine alle Elemente des synchronen polyadischen π-Kalküls unterstützt, lassen sich beliebige π-Prozesse, die die Syntax aus Tabelle 2.3 nutzen, durch die virtuelle Maschine ausführen. Komplexere Konstrukte wie if-then-else werden unterstützt, wenn sie sich auf den synchronen π-Kalkül abbilden lassen. Daraus ergeben sich für die Erweiterung der Funktionalität der virtuellen Maschine verschiedene Ansätze:

- Es können Funktionen über das τ eingebunden und aus Prozessen heraus aufgerufen werden.
- Ein Compiler kann komplexe Konstrukte auf die von der virtuellen Maschine unterstützten Konstrukte abbilden.
- Die Behandlung neuer Konstrukte kann innerhalb der virtuellen Maschine implementiert werden.

5.5.1 Weiterentwicklung der virtuellen Maschine

Die hier vorgestellte virtuelle Maschine ist nur ein Prototyp und noch nicht für den Produktiveinsatz geeignet. Neben der Steigerung der Geschwindigkeit sind noch einige funktionale Forderungen umzusetzen, die in diesem Abschnitt beschrieben werden sollen.

Fehlerbehandlung

Für den Einsatz als Ausführungsmaschine für Geschäftsprozesse muss es eine Möglichkeit zur Fehlerbehebung geben. Die von BPEL bekannten CompensationHandler können als einfache π-Prozesse modelliert werden, deren Ausführung erst nach Beendigung eines regulären Prozesses ermöglicht wird. Der Prozess in Gleichung 5.3 erzeugt nach der eigentlichen Prozesserfüllung (hier als τ abstrahiert) einen Kanal $comp$, den er mit der Beendigung des Prozesses an den Client schickt. Gleichzeitig wird ein Kompensierungsprozess abgespalten, der über den Kanal $comp$ angestoßen werden kann.

$$A(uri) \stackrel{def}{=} !uri(o, req).\tau.\mathbf{v}comp\ (\overline{o}\langle comp\rangle.\mathbf{0} \mid comp.C) \tag{5.3}$$

Schwieriger ist die Gestaltung von FaultHandlern, die nicht ohne Mitwirkung der virtuellen Maschine realisierbar sind. Eine Möglichkeit ist, einen Namen in der virtuellen Maschine zur Fehlerbehandlung zu reservieren. Dies könnte beispielsweise der URI *urn:pivm.org:error* sein, im folgenden als e bezeichnet. Jedes Mal, wenn ein Fehler auftritt, sendet die virtuelle Maschine an diesen Namen eine Fehlermeldung und ein Prozess, der von diesem Namen liest, kann den Fehler behandeln. Damit jeder Prozess seinen eigenen FaultHandler definieren kann, wird die Namensersetzung verwendet. Zur Definition eines neuen FaultHandler erzeugt der Prozess den Namen e erneut, was zur Folge hat, dass ein neuer Name in die Ersetzungstabelle des Prozesses eingetragen wird. Tritt ein Fehler in einem Prozess P auf, schlägt die virtuelle Maschine die aktuelle Ersetzung des Namens e in der Ersetzungstabelle von P nach und schickt an diesen Namen die Fehlermeldung.

$$P \stackrel{def}{=} \mathbf{v}e\ (e(message).E(message) \mid P') \tag{5.4}$$

Hier definiert der Prozess P seinen eigenen Fehlerkanal e. Tritt innerhalb von P' ein Fehler auf, sendet die virtuelle Maschine Fehlermeldungen an das restringierte e und der Prozess E wird angestoßen. Der fehlerhafte Prozess P' wird von der virtuellen Maschine terminiert.

Innerhalb von P' kann der Fehlerkanal wieder ‚überschrieben' werden. Im Unterschied zu den FaultHandlern aus BPEL und zu Exceptionhandlern aus objektorientierten Sprachen wie Java sind die hier vorgestellten Fehlerkanäle nicht verschachtelt. Bei der Abarbeitung von π-Prozessen wird kein Stack benötigt, da es keine Rücksprünge gibt [42] und somit weiß die virtuelle Maschine auch nicht (sie braucht es auch nicht zu wissen), von welchem Prozess ein Agent aufgerufen wurde. Ein Fehlerbehandlungsprozess kann also eine Fehlermeldung nicht erneut werfen, damit sie von einem ihn umschließenden FaultHandler behandelt werden kann. Nichtsdestotrotz lassen sich solche Beziehungen explizit modellieren, wenn sie benötigt werden. Da der Fehlerbehandlungsprozess E als Agent definiert wurde, kann er von jedem beliebigen Prozess heraus aufgerufen werden.

Garbage Collection

Durch die Aufspaltung eines Prozesses in mehrere nebenläufige Prozesse kann es vorkommen, dass einige Prozesse mangels Kommunikationspartner nicht weiter bearbeitet werden können.

$$SYS \stackrel{def}{=} \mathbf{v}(x,y)\ (P(x,y) \mid Q(x,y)) \tag{5.5}$$

$$P(x,y) \stackrel{def}{=} x(a).A(a) \mid y(b).B(b) \tag{5.6}$$

$$Q(x,y) \stackrel{def}{=} \mathbf{v}z\ (\overline{x}\langle z\rangle.\mathbf{0} + \overline{y}\langle z\rangle.\mathbf{0}) \tag{5.7}$$

Im abgebildeten Beispiel kann nur einer der beiden parallelen Prozesse in P mit Q kommunizieren, der andere bleibt als Müll zurück, da durch die Eingrenzung des Geltungsbereiches der Namen x und y kein anderer Prozess darüber Nachrichten senden kann. Ohne ein Verfahren, diesen Müll zu beseitigen (*Garbage Collection*), würde nach einer längeren Laufzeit der dafür benötigte Speicher stark anwachsen und zu Engpässen im System führen. Peschanski und Hym [42] beschreiben ein Algorithmus, der die Identifizierung von solchen *isolierten* Prozessen ermöglicht.

Dazu werden die Referenzen auf restringierte Namen lokal für jeden Prozess und global für die gesamte virtuelle Maschine gezählt. Bei jeder Namensersetzung, die durch den Aufruf definierter Agenten und dem Restringieren oder dem Lesen von Namen verursacht werden können, wird der Zähler des neuen Namens sowohl lokal für den Prozess als auch global um eins erhöht, während der Zähler des vorherigen Namens um eins verringert wird. So weiß die virtuelle Maschine, wie oft ein konkreter Name insgesamt verwendet wird und jeder Prozess weiß, wie oft er ihn verwendet. Wenn die lokale Referenzzahl eines Namens gleich der globalen Referenzzahl ist, kennt kein weiterer Prozess diesen Namen und eine Kommunikation über ihn ist nicht möglich. Eine solche Kommunikation nennt man *gesperrt*. Prozesse, die nur auf gesperrte Kommunikationen warten, sind isoliert und können beseitigt werden.

Kapitel 6

Verwandte Arbeiten

In der Wirtschaft ist die Serviceorientierung mit SOAP-basierten Web-Services momentan weit verbreitet. Dabei hat sich zur Ausführung von Prozessen der BPEL-Standard etabliert. Der Standard ermöglicht das Beschreiben und Ausführen von Web-Service-Kompositionen. Doch öffnen sich die Hersteller von BPEL-Engines allmählich dem neuen Trend und gestatten in ihren Produkten den Aufruf von Ressourcen parallel zu Webservice-Aufrufen oder planen dies für die Zukunft [14, 7]. Man darf also gespannt sein, wie weit sich die Ressourcenorientierung im Unternehmenseinsatz ausbreiten wird.

Auf der anderen Seite beschäftigen sich Arbeiten mit der Nutzbarkeit des π-Kalküls für das Business Process Management. Die π 4 Technologie Foundation [43] beschäftigt sich mit der Validierung von Webservice-Choreographien durch den π-Kalkül und entwickelt in diesem Rahmen das WS-CDL-Framework pi4soa.

Der BPM-Lehrstuhl am Hasso-Plattner-Institut setzt sich mit der Verwendbarkeit des π-Kalküls für die Verifizierung und Implementierung von Geschäftsprozessen auseinander [47]. Puhlmann zeigt in [49] erstmals die Umsetzung der bekannten Workflowpattern durch den π-Kalkül. Bog beschreibt in [11] einen Simulator, der die Ausführung von π-Prozesse visualisiert.

Für die eigentlichen Ausführung von π-Prozessen wurde mit der Programmiersprache PICT [44] experimentiert und mit der CubeVM [42] gibt es bereits eine frühe Version einer virtuellen Maschine für den π-Kalkül. Der Fokus bei diesen Projekten liegt jedoch weniger auf der Ausführung von Geschäftsprozessen in ei-

ner verteilten Umgebung. Es geht vorrangig darum, den π-Kalkül als objektorientierte Sprache zu nutzen.

Kapitel 7

Schlussfolgerungen

Das Ziel dieser Arbeit war es, herauszufinden, ob der π-Kalkül für die Implementierung von Ressourcen im Sinne von REST geeignet ist. Um sich einer Lösung zu dieser Fragestellung zu nähern, wurden in Kapitel 2 die benötigten Technologien eingeführt. In Kapitel 3 wurde anschließend verdeutlicht, wie REST-basierte Applikationen gestaltet werden können und wie sich daraus eine höhere Skalierbarkeit des Systems ergibt. Die Hauptpunkte dabei sind die größtmögliche Entlastung der Server (Caching, clientseitige Sitzungsverwaltung, Code on Demand ...) und ein Scaling-out-Konzept zur Realisierung der Skalierbarkeit bereitzustellen. Dafür wurde der Ansatz der portablen Ressourcen vorgestellt, der es ermöglicht, Ressourcenimplementierungen von einem Server auf einen anderen zu verlagern. Das geschieht transparent gegenüber den Clients, da die Infrastruktur die Anfragen an den URI einer verlagerten Ressource an den neuen Server leitet. Somit kann auf Ressourcenebene eine Lastverteilung implementiert werden.

Wie sich die Kommunikation im Internet über HTTP auf den π-Kalkül abbilden lässt, war Thema des 4. Kapitels. URIs können als π-Namen verwendet werden. Jedoch ist darauf zu achten, dass die Eindeutigkeit der URIs bei deren Erzeugung sichergestellt wird. Die blockierende Kommunikationssemantik des π-Kalküls kann auch im Internet nachgebildet werden. Die virtuelle Maschine simuliert gegenüber den π-Prozessen die Blockierung der Kommunikation während sie wiederholt versucht, mit Ressourcen im Netz zu kommunizieren. Außerdem wurde gezeigt, wie funktionale Aspekte in den prozessorientierten π-Kalkül ein-

gebunden werden können, auch wenn diese Funktionalitäten in anderen Sprachen implementiert wurden. Die Theorie des π-Kalküls ermöglicht ferner, π-Prozesse vor der Verwendung als Ressourcenimplementierungen auf sicheres und idempotentes Verhalten hin zu untersuchen.

Um die theoretischen Erkenntnisse dieser Arbeit zu evaluieren, wurde ein Prototyp einer virtuellen Maschine entwickelt, und in Kapitel 5 vorgestellt. Die virtuelle Maschine nimmt HTTP-Anfragen von Clients entgegen und führt daraufhin π-Prozesse aus. Das Ergebnis der Prozessausführung wird als Antwort an die Clients zurückgesendet. Dabei werden alle π-Prozesse unterstützt, die die in Tabelle 2.3 beschriebene Grammatik nutzen, inklusive der Erweiterungen aus Kapitel 4.

Da die virtuelle Maschine funktionstüchtig ist, wie es auch in einer Demonstration am Hasso-Plattner-Institut in Potsdam gezeigt wurde, ist es möglich, Ressourcen durch den π-Kalkül zu implementieren. Die Hauptfrage dieser Arbeit kann somit bejaht werden: Der π-Kalkül ist Grundsätzlich als Implementierungssprache für Ressourcen geeignet.

Diese Erkenntnis trägt dazu bei, die Entwicklung eines Business-Process-Management-Systems zu ermöglichen, wie es zu Anfang der Arbeit in der Vision erläutert wurde. Graphisch modellierte Geschäftsprozesse werden automatisiert in den π-Kalkül überführt und formal verifiziert. Anschließend können sie auf einen Server deployt und in einer virtuellen Maschine ausgeführt werden. Dabei wird die Semantik des π-Kalküls respektiert und somit behalten die formalen Aussagen der Verifikation ihre Gültigkeit. Gleichzeitig können die Vorteile einer ressourcenorientierten Umgebung genutzt werden.

Als nächsten Schritt auf diesem Weg ist die Erweiterung der virtuellen Maschine und des π-Compilers für Prozessmodelle zu forcieren, die sich beide noch im prototypischen Status befinden. Es wird noch ein Deployment-Konzept für π-Prozesse auf einen Server mit der virtuellen Maschine zu erstellen sein und die Anwendungen müssen zu leicht zu installierenden Paketen zusammengefasst werden. Für den angestrebten Anwendungsfall ist außerdem die Integration in Prozessmodellierungswerkzeugen notwendig.

Wird die Entwicklung in diese Richtung weiterhin aktiv vorangetrieben, so könnte die Vision aus der Einleitung schon in einigen wenigen Jahren realität werden.

Anhang A

Beispielanwendung

Als Beispielanwendung wurde im Rahmen dieser Arbeit ein einfacher Broker realisiert. Der Broker verwaltet mehrere Banken, die sich bei ihm registrieren und deregistrieren können. Ein Kunde kann den Broker nach einer geeigneten Bank fragen, woraufhin der Broker eine aussucht und an den Kunden schickt. Da das Beispiel hauptsächlich zu Demonstrationszwecken dient, ist die verwendete Logik sehr einfach gehalten. Der Broker gibt immer die Bank zurück, die sich zuletzt bei ihm registriert hat.

Der Broker speichert alle registrierten Banken in einer Datenbanktabelle. Banken können sich über eine HTTP-POST-Nachricht registrieren oder deregistrieren. Dazu muss der Inhalt der Nachricht die Felder `bank` und `method` enthalten, wobei `bank` als Wert ein URI zur Bank enthalten soll und die Methode muss entweder `register` oder `deregister` enthalten. Ein gültiger Nachrichteninhalt wäre somit:

```
bank=http://sparkasse.de&method=register
```

Das Ein- und Austragen einer Bank in die/aus der Datenbanktabelle erfüllen die Methoden `register` und `deregister`. Beide Methoden benötigen als Eingabeparameter den URI der Bank und geben einen Statuscode und eine Antwort im HTML-Format zurück.

Fragt ein Kunde vom Broker eine Bank an, so sendet er einfach eine HTTP-GET-Nachricht an ihn. Der Broker ruft die Funktion `getbank` auf, die den URI einer Bank aus der Tabelle zurückgibt. Die Methode `response` führt diesen Namen in eine HTML-Antwort ein und gibt außerdem einen HTTP-Statuscode zurück.

Der komplette π-Prozess des Brokers ist hier in einem Format dargestellt, wie ihn auch die Mobility Workbench [57] nutzt. Der Restriktionsoperator wird als ^ und Schreibeoperationen als 'x<y> dargestellt. Agenten werden zeilenweise mit agent A(x,y)= definiert und ein t ist ein τ.

```
agent Broker(uri)=!uri(o,req,props,body).
  (^retc)'req<http/REQUEST_METHOD,retc>.retc(ret).
  ([ret=http/GET]t.
  BrokerGet(o)+[ret=http/POST]t.BrokerPost(o,req,body))
agent BrokerGet(o)=t{getbank}(bank).
  t{response}<bank>(code,response).
  'o<code,response>.0
agent BrokerPost(o,req,body)=(^retc)'body<method,retc>.
  retc(method).'body<bank,retc>.retc(bank_uri).
  ([method=register]t.
  BrokerRegister(o,bank_uri) + [method=deregister]t.
  BrokerDeregister(o,bank_uri))
agent BrokerRegister(o,bank_uri)=
  t{register}<bank_uri>(code,response).
  'o<code,response>.0
agent BrokerDeregister(o,bank_uri)=
  t{deregister}<bank_uri>(code,response).
  'o<code,response>.0
```

Literaturverzeichnis

[1] W. M. P. van der Aalst: *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, 8(1):21--66, 1998. http://citeseer.ist.psu.edu/vanderaalst98application.html.

[2] W. M. P. van der Aalst, A. H. M. ter Hofstede, A. P. Barros und B. Kieposzewski: *Workflow Pattern*. Distributed and Parallel Databases, 14(1):5--51, Jan. 2003.

[3] S. Aier und M. Schönherr (Hrsg.): *Enterprise Application Integration - Serviceorientierung und nachhaltige Architekturen*. Gito, 2. Aufl., Juli 2006.

[4] Amazon.com: *Amazon Web Services*. http://www.amazon.com/aws, besucht: 13. Juli 2008.

[5] Amazon.de GmbH: *Amazon.com und Amazon.de melden Rekord-Weihnachtsgeschäft*. Pressemitteilung. www.presseportal.ch/de/text/story.htx?nr=100522171, besucht: 13. November 2007.

[6] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic und S. Weerawarana: *Business Process Execution Language for Web Services Version 1.1*. Standard, Mai 2003.

[7] Apache Software Foundation: *RESTful BPEL Part I + II*. Webseite. http://ode.apache.org/restful-bpel-part-i.html, besucht: 3. März 2008.

[8] T. Bellwood, L. Clément und C. von Riegen: *UDDI Version 3*. Spezifikation, OASIS, Okt. 2003. http://uddi.org/pubs/uddi-v3.0.1-20031014.htm.

[9] T. Berners-Lee: *WWW: Past, present, and future*. In: *IEEE Computer*, Bd. 29, S. 69--77. Okt. 1996.

[10] T. Berners-Lee *et al.*: *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986, W3C, Jan. 2005.

[11] A. Bog: *A Visual Environment for the Simulation of Business Processes based on the Pi-Calculus*. Diplomarbeit, Hasso-Plattner-Institute for IT Systems Engineering at the University of Potsdam, Okt. 2006.

[12] D. Booth und C. K. Liu: *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, 2007. `http://www.w3.org/TR/wsdl20-primer`.

[13] S. Brias: *ABC Bisimulation Checker*, 2005. `http://lamp.epfl.ch/~sbriais/abc/abc.html`, besucht: 24. Juni 2008.

[14] Cape Clear: *On-Demand Integration*. White paper, Sep. 2007. `http://www.capeclear.com/pdf/whitepapers/On_Demand_IntegrationWhitePaperv75-2_dc.pdf`, besucht: 3. März 2008.

[15] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana: *Web Services Description Language (WSDL) 1.1*, 2001. `http://www.w3.org/TR/wsdl`, besucht: 5. Juli 2008.

[16] I. D. Craig: *Virtual Mashines*. Springer-Verlag, Berlin, Sep. 2005.

[17] *DENIC*. `http://www.denic.de/`, besucht: 17. Juli 2008.

[18] O. Döhring, S. Golega, W. Lindenthal, O. Märker, L. Neitsch, G. Schmidt und J. Schulz-Hofen: *Magrathea: Concepts and results in the context of PESOA*. Techn. Ber., HPI, März 2006.

[19] C. Ferris, K. Lawrence und T. Storey: *IBM Submission for the W3C Workshop on Web of Services for Enterprise Computing*. Techn. Ber., W3C, 2007. `http://www.w3.org/2007/01/wos-papers/ibm`.

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

[21] R. T. Fielding *et al.*: *Hypertext Transfer Protocol - HTTP/1.1*. RFC 2616, W3C, Juni 1999.

[22] M. Fowler: *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman, Amsterdam, Nov. 2002.

[23] Google: *Google Data APIs*. `http://code.google.com/apis/gdata/`, besucht: 13. Juli 2008.

[24] E. R. Harold: *REST vs. WS-*: A Parable*, Mai 2006. `http://cafe.elharo.com/web/rest-vs-soap-a-parable/`, besucht: 3. Mai 2008.

[25] P. Helland: *Life beyond Distributed Transactions: an Apostate's Opinion*. In: *Proceedings of the 2007 CIDR Conference*, S. 132--141, Jan. 2007.

[26] M. Henning: *The Rise and Fall of CORBA*. ACM Queue, 4(5):28--34, Juni 2006.

[27] IBM: *Web Services architecture overview*. Techn. Ber., 2000.

[28] International Organization for Standardization: *Database Language SQL*. ISO-Standard, 1992.

[29] *Javascript*. `http://developer.mozilla.org/en/docs/JavaScript`, besucht: 13. Mai 2008.

[30] D. Jordan, J. Evdemon *et al.*: *Web Services Business Process Execution Language Version 2.0*. Spezifikation, OASIS, Apr. 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[31] L. Lamport: *Paxos Made Simple*, Nov. 2001.

[32] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown und R. Metz: *Reference Model for Service Oriented Architecture*. Techn. Ber. 1.0, OASIS, Aug. 2006. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm`.

[33] Microsoft: *ASP.NET*. `http://asp.net`, besucht: 13. November 2007.

[34] R. Milner: *The Polyadic π-Calculus: A Tutorial*. In: F. L. Bauer, W. Brauer und H. Schwichtenberg (Hrsg.): *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktoberdorf, Germany, 1991)*, Bd. 94 d. Reihe *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[35] R. Milner: *Communicating and Mobile Systems: the π-Calculus*. Cambridge Univ. Press, Cambridge, 1999.

[36] R. Milner, J. Parrow und D. Walker: *A Calculus of Mobile Processes, Part I/II*. Journal of Information and Computation, 100:1--77, Sep. 1992. `http://www.lfcs.informatics.ed.ac.uk/reports/89/ECS-LFCS-89-85/`.

[37] P. Mockapetris: *Domain Names - Concepts and Facilities*. RFC 1034, Information Sciences Institute University of Southern California, 1987.

[38] *Mongrel HTTP-Bibliothek und -Server*. `http://mongrel.rubyforge.org/`, besucht: 14. März 2008.

[39] H. Overdick: *Xenodot - XML management*. Foliensatz, Mai 2006. `http://bpt.hpi.uni-potsdam.de/pub/Public/HagenOverdick/xenodot_introduction.pdf`, besucht: 16. Juli 2008.

[40] H. Overdick: *The Resource-oriented Architecture*. In: *2007 IEEE Congress on Services*, S. 340--347, Juli 2007.

[41] H. Overdick, F. Puhlmann und M. Weske: *Towards a Formal Model for Agile Service Discovery and Integration*. In: *Proceedings of the ICSOC Workshop on Dynamic Web Processes (DWP 2005)*, Amsterdam, Netherlands, Dez. 2005.

[42] F. Peschanski und S. Hym: *A Stackless Runtime Environment for a Pi-calculus*. Techn. Ber., Université Pierre et Marie Curie - Paris 6 - LIP6 and Université Denis Diderot - Paris 7 - PPS, Okt. 2006.

[43] *Pi4Tech*. www.pi4tech.org.

[44] B. C. Pierce und D. N. Turner: *Pict: A Programming Language Based on the Pi-Calculus*. In: G. Plotkin, C. Stirling und M. Tofte (Hrsg.): *Proof, Language and Interaction: Essays in Honour of Robin Milner*, S. 455--494. MIT Press, 2000.

[45] J. Postel und other: *Internet Protocol*. RFC 791, Information Sciences Institute University of Southern California, 1981.

[46] J. Postel und other: *Transmission Control Protocol*. RFC 793, Information Sciences Institute University of Southern California, 1981.

[47] F. Puhlmann: *Why do we actually need the Pi-Calculus for Business Process Management?* In: *Proceedings of the 9th International Conference on Business Information Systems*, Klagenfurt, Austria, 2006. to appear.

[48] F. Puhlmann: *On the Application of a Theory for Mobile Systems to Business Process Management*. Doctoral thesis, University of Potsdam, Potsdam, Germany, Juli 2007.

[49] F. Puhlmann und M. Weske: *Using the Pi-Calculus for Formalizing Workflow Patterns*. In: W. M. P. van der Aalst et al. (Hrsg.): *Business Process Management*, Bd. 3649 d. Reihe *LNCS*, S. 153--168. Springer-Verlag, Berlin, Dez. 2005.

[50] *Ruby*. `http://www.ruby-lang.org`, besucht: 2. März 2008.

[51] *Ruby On Rails*. `http://www.rubyonrails.org`, besucht: 13. November 2007.

[52] D. Sangiorgi und D. Walker: *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[53] W. Schuster: *The Futures of Ruby Threading*. InfoQ, Mai 2007. `www.infoq.com/news/2007/05/ruby-threading-futures`, besucht: 14. März 2008.

[54] *Slashdot Nachrichtenseite*. `http://slashdot.org`, besucht: 11. Mai 2008.

[55] *Squid Caching Proxy*. http://www.squid-cache.org/, besucht: 13. Mai 2008.

[56] Sun Microsystems: *JavaServer Faces*. http://java.sun.com/javaee/javaserverfaces, besucht: 13. November 2007.

[57] B. Victor und F. Moller: *The Mobility Workbench --- A Tool for the π-Calculus*. In: D. Dill (Hrsg.): *CAV'94: Computer Aided Verification*, Bd. 818 d. Reihe *Lecture Notes in Computer Science*, S. 428--440. Springer-Verlag, 1994.

[58] W3C: *Simple Object Access Protocol (SOAP) Version 1.1*. Spezifikation, W3C, Mai 2000. http://www.w3.org/TR/soap11.

[59] W3C: *SOAP Version 1.2*. Spezifikation, W3C, Apr. 2007. http://www.w3.org/TR/soap12.

[60] M. Weske: *PESOA: Process Family Engineering in Service Oriented Applications*. Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, Juli 2004.

[61] M. Weske: *Business Process Management*. Springer-Verlag, 2007.

[62] WHATWG: *HTML5 Specification Working Draft*, Nov. 2007.

[63] S. A. White: *Business Process Modeling Notation (PBMN)*. BPMI Specification, Mai 2004.

[64] World Wide Web Consortium: *Document Object Model (DOM)*. http://www.w3.org/DOM/, besucht: 13. August 2008.

[65] World Wide Web Consortium: *XML Path Language (XPath) 2.0*. http://www.w3.org/TR/xpath20/, besucht: 13. August 2008.

Glossar

B2B Business to Business. 7

B2C Business to Customer. 7

BPEL Business Process Execution Language. 12, 51, 82

BPMN Business Process Modeling Notation. 42, 50, 51

Content-Negotiation Client und Server einigen sich auf ein bestimmtes Format für eine Information. 44, 45

DOM Document Object Model. 79

HTML Hypertext Markup Language. 81

HTTP Hypertext Transfer Protocol. 15, 53

IP Internet Protocol. 53

LTS Labeled Transition System. 29

MDD Model Driven Development. 50

PESOA Process Family Engineering in Service-Oriented Applications. 50

REST Representational State Transfer. 2, 21, 51, 65, 69, 87

SCS Client-side session and persistent storage. 41

SOA service-orientierte Architektur. 3, 21

TCP Transmission Control Protocol. 53, 60

UDDI Universal Description, Discovery and Integration. 11

URI Uniform Resource Identifier. 16, 53, 87

WSDL Web Services Description Language. 11, 51

WWW World Wide Web. 20

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Olaf Märker