Simulation von Web-Services

Seminararbeit
von
H. Alvandi Mohajerani

vorgelegt am
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Fakultät für Mathematik & Informatik
Universität Mannheim

Juli 2004

Inhaltsverzeichnis

Abkürzungsverzeichnis

SCET:	Service Composition and Execution Tool
QoS:	Quality os Service
WSDL:	Web Service Description Language
WSFL:	Web Service Flow Language

1 Einleitung

Die neue Generation weltweiter Geschäftsabwicklungen sind die Web-Services. Sie sind die zugänglichen Softwarebausteine des Netzes. Aber es gibt manchmal Probleme bei der Beschränkung ihre Funktionalität, wenn diese tatsächlich eingesetzt werden müssen. Diese Beschränkung kann überwunden werden, indem man Web Prozesse erzeugt. Sie werden erzeugt, indem man mehrere Web-Services kombiniert, um neue Services zu erzeugen. Für die Kunden wird die Fähigkeit eines Web-Services nach verschiedenen Kriterien bewertet. Deshalb ist es notwendig für die Kunden, eine Art und Weise des Auswertens der Services festzulegen. Wir stellen eine Methode vor, die die Dienstgüte (Quality of Service /QoS) eines Web-Prozesses, basierend auf den Attributen der einzelnen existierenden Services, berechnet. Das Modell basiert auf drei Kriterien: Zeit, Kosten und Zuverlässigkeit. Wenn die Ausführungszeit des Web Prozesses nicht die spezifizierten QoS-Anforderungen einhalten kann, muss dieses Problem durch einige Änderungen behoben werden. Um festzustellen, welche Änderungen getätigt werden müssen, benutzt man Simulationsprogramme, die sogenannten „What if" Fragen auf eine kostengünstige und schnelle Art beantworten können.
In dieser Arbeit wird das „Service Composition and Execution Tool" (*SCET*), und die verschiedenen Methoden, die für das Auswerten der Leistung eines NetzProzesses angewendet werden können, beschreiben. Hierbei wird der JSIM–Simulator als eine Auswertungsmethode eingesetzt.

2 Beispiel - Szenario

Unser Beispiel-Szenario[1] stellt eine Buchtransaktion mit der Amazon Web-Service Schnittstelle dar. Wie in Abbildung 1 zu sehen, sind *BarnesGetPrice*, *CheckCredit*, *CheckInventory*, *GenerateBackOrder*, *ReleaseOrder* und *SendCreditLowInfo* die möglichen Aktivitäten unseres Szenarios.
Der Preis des vom Kunden gewählten Buches, wird mit dem *BarnesGetPrice-Service* abgefragt. Das Benutzerkonto wird dann nach ausreichendem Kredit mit dem *CheckCredit-Service* geprüft. Der CheckCredit-Rückgabewert liefert Informationen darüber, ob das Benutzerkonto einen ausreichenden Kredit aufweist. Ist dies nicht der Fall, wird der *SendCreditLowInfo-Service* aufgerufen, ansonsten der *CheckInventory-Service*. Wenn der CheckInventory eine Bestätigung zurückgibt, wird der *ReleaseOrder-Service* aufgerufen, um das Buch zu bestellen, andernfalls wird der *GenerateBackOrder-Service* aufgerufen für die Stornierung der Bestellung.

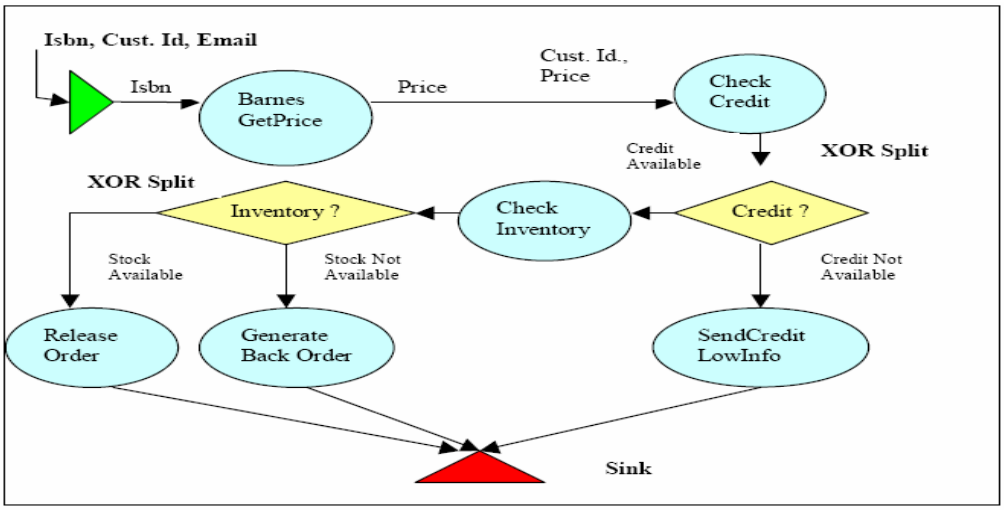

Abbildung 1- Szenario[1]

3 Architektur -Übersicht

In diesem Abschnitt wird eine Einführung in eine Systemarchitektur gegeben, damit die Zusammenhänge im Detail anhand des Buchhandlung - Szenarios, und die Aufgabe des *JSIM* in Zusammenhang mit SCET[1] klar werden.
Abbildung 2 zeigt die Architektur unseres Systems. Wie man sieht, sind die Hauptmodule unseres Systems das „Service Composition and Execution Tool“ *(SCET)*, der JSIM-Simulator und der *Perl* Execution-Controller. Wobei SCET selbst aus einem „Model Generator”, einem „Perl Code Generator” und einem „Execution Monitor” als Submodule besteht. Diese werden in den folgenden Abschnitten definiert.

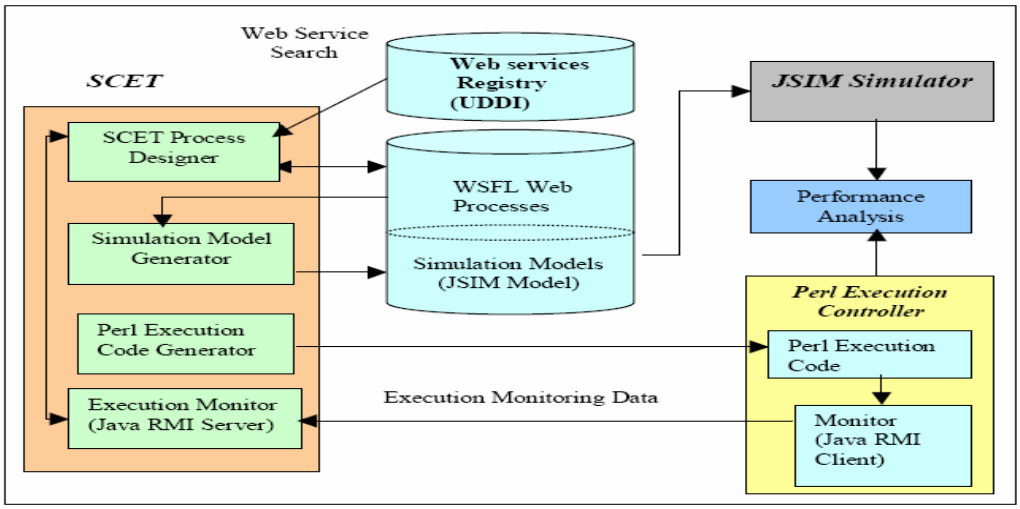

Abbildung 2 – System Architektur vom SCET[1]

3-1 SCET Process Designer

Der SCET Process Designer stellt einem Benutzer ein grafisches Designwerkzeug zur Verfügung, womit Prozesse statistisch erzeugt werden können. Der Web-Prozess wird grafisch durch „Source-Knoten“, „Sink-Knoten“, „Activity-Knoten“, „Data-Links“ und „Control-Links“ (Abbildung 3) repräsentiert. Die schwarzen Links in der Abbildung stellen Control-Links dar. die Conrol-Links die Control-Flow-Bedingungen (also die Reihenfolge, in der der Code durchlaufen wird) für das Netz spezifizieren, während die Data-Links

bestimmen, welcher Ausgang einer Aktivität auf welchen Eingang einer anderen Aktivität umgeleitet wird.
Ein Activity-Knoten speichert Informationen über den ausführenden Web-Service. Dieser beinhaltet die Position der WSDL-Datei, der Web-Services und QoS-Informationen, die entweder vom Service Provider oder aus dem Analysetest hervorgehen (z.B. Mean Service Time, Verfügbarkeitsfaktor, uzw.). Diese werden für den JSIM-Simulator verwendet, um das Prozessverhalten zu simulieren. Zum Schluss kann der Designer den Prozess als WSFL-basierte Spezifikationsdatei für die Weiterverarbeitung speichern.

3-2 Simulation Model Generator

Der in unserem System benutzte JSIM-Simulator erfordert eine Java basierte Spezifikation des zu simulierenden Modells [1,3]. In SCET stellen wir einen bestehenden Prozess als WSFL basierte Spezifikation dar. So müssen wir diese WSFL-Daten in eine Form umwandeln, die an den JSIM-Simulator angepasst werden kann. Der Simulation-Model-Generator von SCET kann diese Aufgabe erledigen.

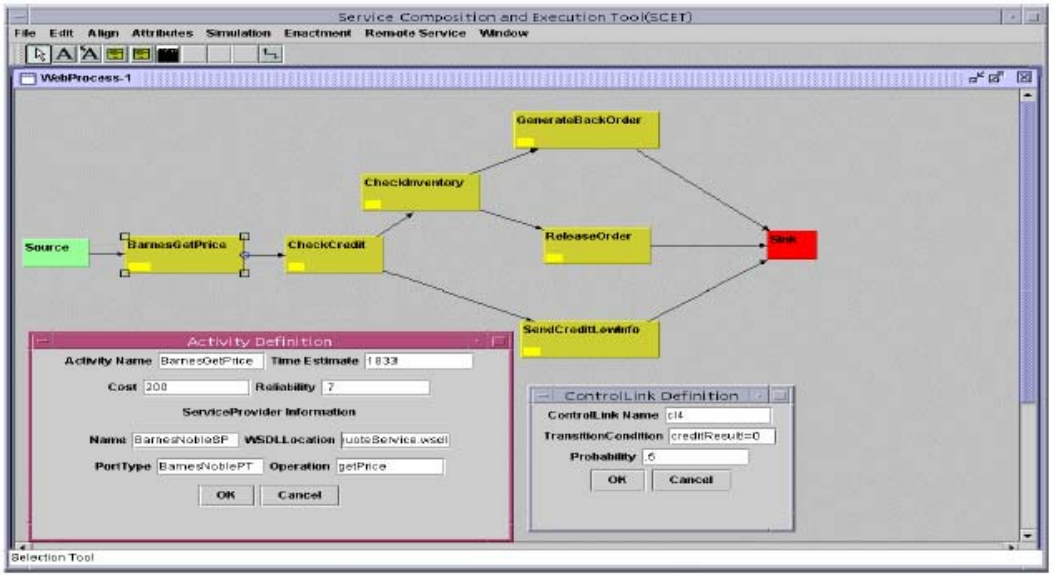

Abbildung 3 - SCET Process Designer[1]

3-3 Perl Execution Code Generator

Dieses SCET-Submodul erzeugt anhand der eingegebenen WSFL – Daten den Perl Execution Code für die Ausführung des Prozesses. Hier werden z.B. die Aktivitätsknoten aus dem Design zu Anforderungscodeblöcken umgewandelt, wie in Abbildung 4 gezeigt wird.

```
my $GetBNPrice = SOAP :: Lite
 ->service('http://www.xmethods.net/sd/2001/BNQuoteService.wsdl');
$GetBNPriceResult = $GetBNPrice->getPrice($isbn);
```
Abbildung 4 – Perl Execution Code

3-4 WSFL Spezification Generator

Wie in den vorherigen Abschnitten erwähnt, speichert SCET den Prozess als WSFL basierte Spezifikation. Die Umwandlung vom intern gespeicherten Modell zur WSFL Spezifikation ist direkt. Die Quell-/Zielknoten im Prozessdesign ergeben Quell-/Zielelemente der WSFL Spezifikation. Die Aktivitätsknoten des Designs werden in Aktivitätselemente der WSFL Spezifikation umgewandelt. Die Input/Output Informationen, die mit jeder Aktivität verbunden sind, werden zu den Ein-/Ausgabeelemente der entsprechenden Aktivitäten in WSFL umgewandelt. Die Links, welche die Knoten im Graph verbinden, sind entweder Control- oder Data-Links. Sie werden ebenfalls in Conrol bzw. Data Link Elemente umgewandelt. Abbildung 5 zeigt eine Struktur des WSFL basierten Codes, der für den „BarnesBookPurchase" Prozess erzeugt wird.

```
<definitions>
<message name = …>
.....
<activity name = …>
…….
<controlLink name="cl6" source="CheckInventory"
        target="GenerateBackOrder"
condition="CheckInventoryResult==0" probability=".2" />
.....
<dataLink name="dl2" source="Search Amazon"
target="CheckInventory">
        <mapInfo sourcePart="bookIsbn"
        targetPart="bookIsbn" />
</dataLink>
</definitions>
```

Abbildung 5 – WSFL Spezifikationscode[1]

3-5 Web Process Execution

SCET ermöglicht die automatischen Erzeugung eines Perl Execution Codes aus WSFL basierten Prozessbeschreibungen. In unserer Implementierung übernimmt ein Perl Controller die gesamte Web-Process-Execution. Der Perl Execution Controller bindet die Aktivitäten mit einem Pipe-und-Filter Modell zusammen, welches den Ausgang eines Web-Service passend auf den Eingang eines anderen Web-Services legt.
Im "BarnesBookPurchase" Web-Prozess rufen wir die Methode „getPrice( )" auf dem „BarnesGetPrice( )" Web-Service mit „bookIsbn" als Stringparameter auf. Der Perl-Code dieser Anwendung ist in Abbildung 4 zu sehen und ermöglicht die Verwirklichung der BarnesGetPrice Aktivität des Prozesses und die Speicherung seines Resultats für spätere Zwecke . Im folgenden Abschnitt besprechen wir Techniken zum Auswerten der Leistung eines bestehenden Prozesses.

4 Performance Evaluation

Die Leistungsbewertung der Web-Services hilft dem Designer, das Verhalten der Aktivitäten in einem bestehenden Prozess zu verstehen und hilft Fehler vor dem Einsatz des Services festzustellen und beheben zu können. Die Ausführungszeit einer einzelnen Web-Service-Anforderung setzt sich aus drei Bestandteilen zusammen: Service Time (*S*), Delay Time (*M*) und Waiting Time (*W*) [1,3].
Service-Time ist die Zeit, die der Web-Service für sich beansprucht, um seine Aufgabe durchzuführen. *Delay-Time* ist die Zeit, die eine SOAP-Nachricht für ihre gesendete/empfangene Anforderungsanrufe benötigt, die u.a. von der Größe der SOAP-Nachricht abhängt. *Waiting-Time* ist die Zeitspanne, die Netzservice-Anforderungen wegen der Systemauslastung warten müssen.
Die gesamte Anforderungszeit (T) für einen Web-Service ist also durch folgende Formel gegeben:

$$T(\sigma) = M(\sigma) + W(\sigma) + S(\sigma)$$

Die zu erwartende Anzahl an Aufrufen eines Services lässt sich durch folgende Formel berechnen:

$$L_n(\sigma) = \frac{W(\sigma_n) + S(\sigma_n)}{\bar{S}_n(\sigma)}$$

Wobei $S_{n\,(\sigma)}$ der Moving-avg.-Service-Time entspricht, die gegeben ist durch:

$$\bar{S}_n(\sigma) = \frac{\sum_{i=1}^{n} (T(\sigma_i) - M(\sigma_i))}{n}$$

Abbildung 6 zeigt die Resultate eines Testdurchlaufes unseres Szenarios. Sie stellt Informationen für unsere Analyse zur Verfügung. Man sieht, dass der „SendLowCreditInfo“ Web-Service eine hohe Wartezeit verursacht. Dies kann bedeuten, dass entweder der Web-Service oder das System, das diesen Service unterstützt, nicht in der Lage ist, die entstehende Last zu ertragen. Durch Ersetzen des SendLowCreditInfo Web-Service durch einen äquivalenten Web-Service, der mehr Netzlast erträgt, könnte man vielleicht dieses Problem beheben.

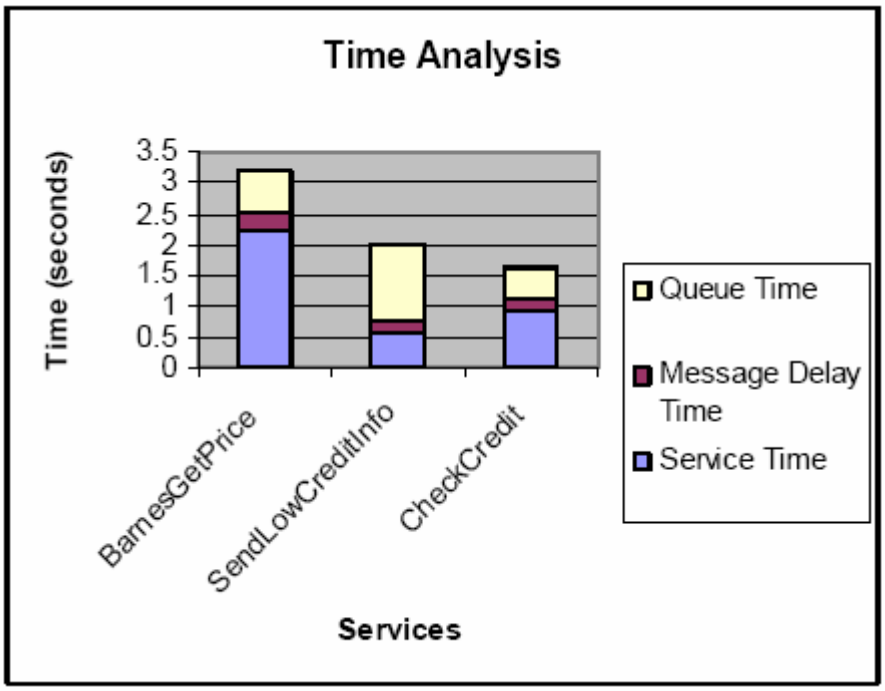

Abbildung 6- Time Analyse[1]

5 Process Controller

Wir benutzen den Java RMI Mechanismus zur Überwachung der Ausführung jeglicher Funktionen in unserem System. Nach der Ausführung der Prozesse enthält der „Perl-Execution Controller“ die Daten. Diese Daten werden einem Java RMI Client übermittelt, der wiederum die überwachten Daten dem Java RMI Server des SCET Designers übermittelt, um dem Benutzer die Visualisierung der Ausführungsdaten zu ermöglichen. Wir haben diese Annäherung in unserem System als Brücke verwendet, um die Informationen zwischen dem „Perl Execution Controller“ und dem „Java-basierten SCET Designer“ auszutauschen.
Diese Ergebnisanalysetechniken stellen ein Qualitätsfeedback der Prozesse zur Verfügung. Da diese Techniken in einer kontrollierten Weise durchgeführt werden, kann man eine gute Schätzung über die Qualität treffen, wenn die betroffenen Web Services unter unserer Kontrolle sind. Solche Probleme passieren, wenn z.B. der Server, der den Web Service unterstützt, mit anderen Programmen schwer ausgelastet ist, und dadurch wird seine Service-Zeit in der Analyse falsch berechnet. Um diese Schwierigkeit zu überwinden, bietet die Simulation der Web-Services eine Lösung mit optimalem Kostenaufwand.

6 JSIM

Der Prozess wird unter Berücksichtigung der Änderungs- bzw. Verbesserungsvorschläge simuliert und vergleicht die Resultaten mit den vom Perl-Code-Execution erzeugten Daten vergleichen. Zum Beispiel:
Ersetzen der Services, die nicht zur Verfügung stehen können, in dem „Avg. Service Time“ durch andere Services.
Da die Funktionalität von JSIM auf Java und Java-Beans basiert, ist JSIM für die Simulation von Web-Services gut einsetzbar und kann sowohl zentralisierte als auch verteilte Web Service Strukturen simulieren.

JSIM enthält drei Fundamental-Pakete: *Queue, Statistic, Variates* **.** Außerdem enthält er noch verschiedene andere Pakete, die im Folgenden detailliert beschrieben werden [2] (Abbildung 7).

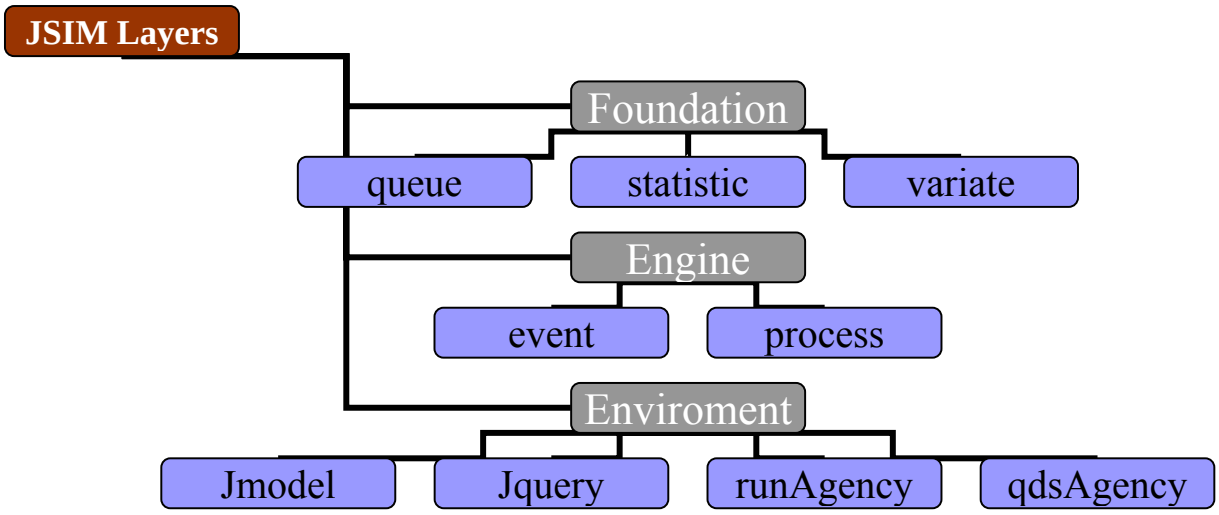

Abbildung 7 JSIM Pakete

6-1 Queue Packet

Dieses Paket definiert eine abstrakte Basis-klasse, der die Klasse LIFO-Queue, FIFO-Queue, Periority-Queue und Temporal-Queue abgeleitet sind, wobei die Temporal und Periority-Queues eine Baumstruktur und die LIFO und FIFO-Queues eine einfache Listenstruktur verwenden.

6-2 Statistic Packet

Das Statistic Packet enthält die Klassen bzw. die klassenzugehörigen Methoden, die für das Analysieren der statistischen Informationen benötigt werden. Die Statistikklasse ist für die Berechnung von Minima, Maxima, Mittelwerten, Varianzen, uzw. der Daten zuständig. Man kann mit diesem Paket zum Beispiel auch statistische Histogramme erzeigen.

6-3 Variate Packet

Die Variante-Klasse, die auch als Random-Generator bezeichnet ist, besitzt beim JSIM einen eigenen linearen Random Generator (LCGRandom). In JSIM existieren Implementierungen von vierzehn kontinuierlichen Wahrscheinlichkeitsverteilungen (Beta, Cauchy, ChiSquare, Erlang, Exponential, F Verteilung, Gamma, HyperExponential, LogNormal, Normal, StudentT, Dreieck und Weibull) sowie acht diskrete(Bernoulli, Hypergeometrisch, Binomial, DiskreteProb, Geometrisches, NegativeBinomial, Poisson und Randi).

6-4 Event Packet

JSIM beinhaltet ein Event Paket, das benutzt werden kann, um das Event-Scheduling zu simulieren. Dieses Paket besteht aus Event-, Entity- und Schedule-Klassen. Die Event-Klasse verwendet man, um bei Erhalt eines Pakets entscheiden zu können, in welchen Zustand gewechselt werden muss. Die Entity-Klasse wird benutzt, um Informationen über Entities (z.B. Kunden) in der Simulation beizubehalten. Schließlich wird die Schedule-Klasse, die Vorgehensweise für zukünftige Events festzulegen, indem sie in einer Event-Liste platziert wird.

6-5 Process Packet

Das Process-Packet liefert alle Klassen, um Simulationsmodelle zu erzeugen, die dem Prozess-Interaktion-Paradigma folgen. Ein Simulationsmodell wird in eine Java-Bean eingekapselt. Solche Bean-Objekte enthalten einige DynamicNodes, wie Server, Facility, Signal, Source und Sink. Diese Knoten werden mit Kanten angeschlossen, die eigentlich als Transportwege der Anforderungen (SimObjects) zwischen den Knoten dienen. Ein Model-Object wird benutzt, um die Simulation zu steuern, sowie Starten und Stoppen der Simulation. Wenn eine Animation durchgeführt werden soll, erzeugt das Modell einen ModelCanvas für die Animation des Projektes.

6-6 QDS Packet

Sowohl die Simulation als auch die Simulationsresultate sind häufig sehr zeit- und rechenintensiv. Folglich ist eine Simulation nur sinnvoll, wenn die Simulationsresultate für einen zukünftigen Gebrauch gespeichert werden. Datenbank-Management-Systeme (DBMSs) eignen sich hierfür aufgrund der Ablage, der Handhabung und der Wiederherstellung der großen Datenmengen besonderes gut. JSIM kann mit einer Vielzahl von DBMS's mittels der Java Datenbank-Schnittstellen (JDBC) verbunden werden. Eine Anfrage versucht zuerst von einem auf QDS (Query Driven Simulation) basierten Simulationssystem die erforderlichen Informationen in der Datenbank herauszusuchen, da sie als Resultat einer früheren Ausführung gespeichert werden konnten.
Sind die erforderlichen Daten vorhanden, werden sie einfach zurückgegeben und dem Benutzer dargestellt. Anderenfalls wird zuvor eine Instanz des entsprechende Modells erzeugt.

6-7 JModel Packet

JSIM bietet eine grafische Oberfläche, die auf Java-AWT basiert. Es erlaubt dem Benutzer ein Simulationsobjekt für das Modell in der richtigen Position zu erzeugen. JModel besteht aus „Source Nodes", „Server Nodes", „Facility Nodes", „Sink Nodes", uzw. (Abbildung 8).

1-Source Nodes: Eine zum „Source-Activity-Node" des SCET-Designers analoge Spezifikation. Mit Hilfe des Random-Variate-Packets, erzeugen sie Anforderungen mit einer „Inter-Arrivel-Time", und dienen als Orientierungspunkt der Simulations-Anforderungen, während die Activity-Nodes als Startpunkte des Prozesses dienen.

2-Server Nodes/Facility Nodes: Sie werden mit einer zufälligen Service-Time, die durch das Random-Variate-Packet erzeugt wird, für die Vorbereitung der benötigten Services eingesetzt. Server haben eine oder mehrere Service-Units ohne Queuing Funktionalität. Facility-Nodes stellen eine Warteschlange für die Anforderungen zur Verfügung. Der Ablauf der Aktivitäten (Web Services) im Prozessdesign hängt bei der JSIM-Modellierung von dem

Verhalten des Queuing-Services ab.

3-Sink Nodes: Sink Nodes rufen Anforderungen auf und speichern Statistikdaten.

4-Transports: Man kann sich die Control-Links beim SCET-Prozess-Designer als Transports im JSIM Simulationsmodell vorstellen. Sie werden für die Verbindung zwischen zwei Knoten benötigt.

5-SimObjects: Als Instanz der Simulations-Anforderungen wird ein SimObjekt erzeugt und die Anforderungen eines Web Prozesses werden wie ein SimObjekt im JSIM Modell dargestellt.

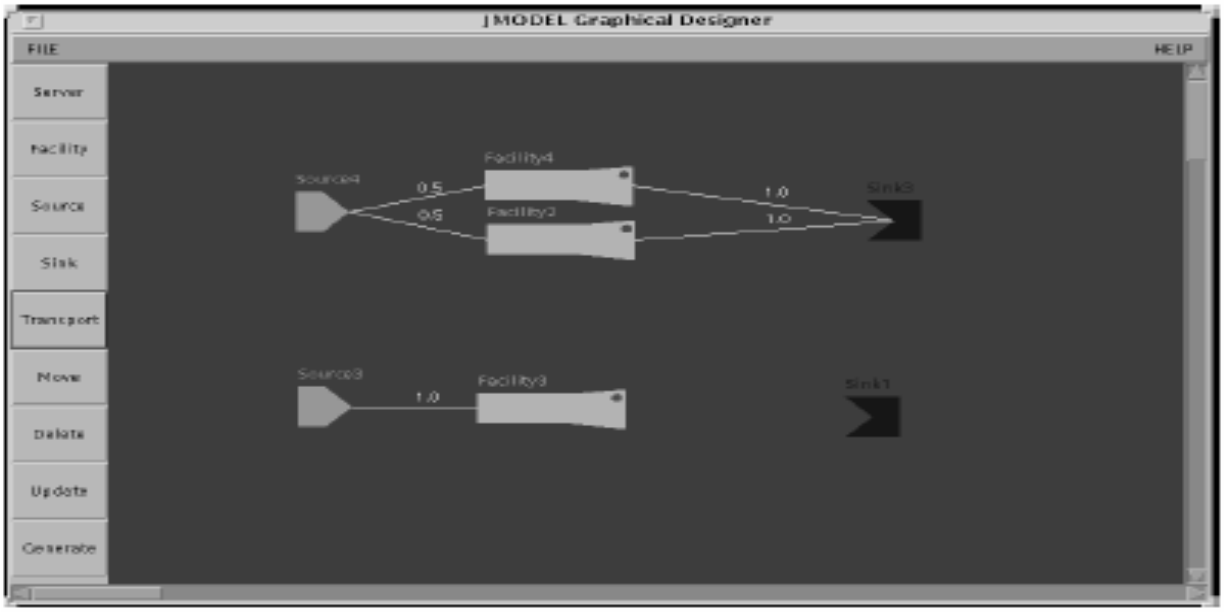

Abbildung 8 Screenshot von JSIM [2]

7 Simulation

Wie schon erwähnt kann JSIM sowohl zentralisierte als auch verteilte Netzstrukturen simulieren. Anwendung auf das Amazon-Szenario zeigen die Abbildung 9 und 10. Wie in diesen Abbildungen zu sehen, ist die Gesamt-Laufzeit bei zentralisierter Struktur größer als bei verteilter Struktur. Der Grund dafür ist, dass bei der zentralisierten Form noch ein Knoten als Controller existiert, was doppelten Aufwand für die Nachrichtenversendung verursacht.

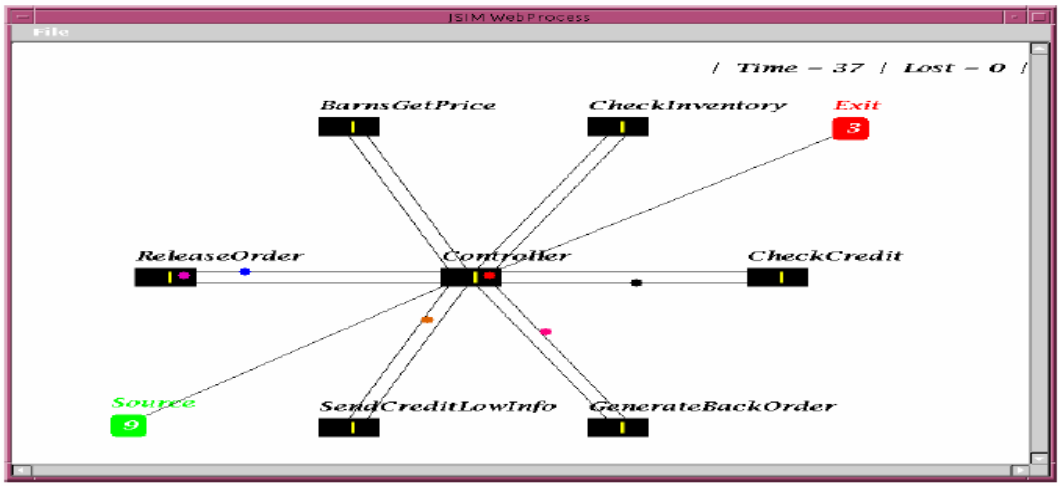

Abbildung 9 JSIM zentralisierter Struktur[1]

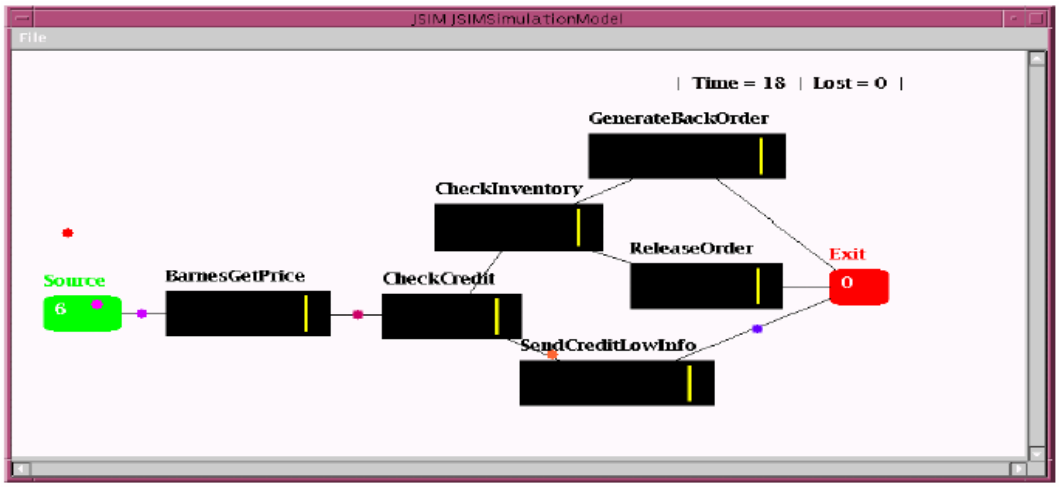

Abbildung 10 JSIM verteilter Struktur[1]

Nachdem der Prozess durch den Prozessdesigner erzeugt wurde, kann der Komponist den Simulationsmodellgenerator verwenden, um die WSFL spezifizierten Daten in JSIM spezifizierte Daten (Java) umzuwandeln. Das JSIM Simulation Model erhält als Eingang die Service-Zeitverteilungsfunktionen, welche die Web-Services charakterisieren. Diese Verteilungsfunktionen können zum Beispiel vom Service-Provider zur Verfügung gestellt werden. Für jeden Control-Link, der in den Prozess miteinbezogen wird, erfordert JSIM einen zugehörigen Wahrscheinlichkeitswert für das Simulieren der Prozessausführung. In unseren Experimenten haben wir die Wahrscheinlichkeitswerte berechnet, die mit jedem Control-Link verbunden sind, indem wir den Prozess auf einer Testgrundlage durchführten. Nach einem Simulationslauf erzeugt JSIM statistische Informationen über die abgeschlossene Simulation. Diese schließen Informationen über Minimum, Maximum, Mittel und Standardabweichung der Zeitabschätzungen aller Aktivitäten ein, die in das Prozessmodell miteinbezogen wurden. Während dieser Simulationsphase kann der Komponist die verschiedenen möglichen Alternativen prüfen, die am Aufbau angewendet werden konnten. Dazu gehört auch das Analysieren, wie Prozesse und einzelnen Web-Services ausgeführt werden müssen, wenn sich die Prozessstruktur ändert. Diese statistischen Informationen geben ein Feedback über das Verhalten des Prozesses für hypothetische Fälle. Die JSIM-Simulation zeigt dynamisch die Anzahl der Anforderungen an, die schon bei einer Aktivität bearbeitet wurden. Der SCET-Designer ermöglicht ebenfalls das Anzeigen der erwarteten Anzahl an realen Anforderungen, die im Host eines Web-Services vorhanden sind. So kann der Prozess zu einem mit JSIM simuliert und zur anderen mit dem Perl-Code

ausgeführt werden. Anschließend können beide Modelle quantitativ miteinander verglichen werden (Tabelle 1, 2). Weichen die Ergebnisse des Simulationsmodells stark von dem ausgeführten Modell ab, können wir Änderungen an den Parametern der Simulation vornehmen, um die Genauigkeit zu verbessern. In unserem Test wurde der tatsächliche Prozess mehrmals mit einer Periode von drei Sekunden wiederholt. Die tatsächlich aufgetretene Avg-Service-Times betrugen 1.833, 862 und 539 Sekunden für BarnesGetPrice, CheckCredit und SendCreditLowInfo. Die Simulationsresultate ergaben Avg-Service-Times von 1.883, 861 und 538 Sekunden. Man sieht also, dass der Simulationsablauf die tatsächliche Ausführung approximiert und unsere Modellierung gut funktioniert.

Statistics

NoSamples	MinValue	MaxValue	MeanValue	Deviation	Interval	Precision	StatName
10.0	3000.0	3000.0	3000.0	0.0	0.0	0.0	Source (dur)
10.0	0.0	0.0	0.0	0.0	0.0		Exit (dur)
10.0	1832.938	1833.028	1833.002	0.025	0.019	0.0	BarnesGetPrice (du
38033.0	0.0	1.0	0.509	0.5	0.005	0.010	BarnesGetPrice (oc
10.0	861.947	862.058	861.997	0.034	0.026	0.0	CheckCredit (dur)
38496.0	0.0	1.0	0.224	0.417	0.004	0.019	CheckCredit (occ)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	CheckInventory (d
0.0	0.0	0.0	0.0	0.0	0.0	0.0	CheckInventory (o
10.0	538.951	539.032	538.991	0.029	0.022	0.0	SendCreditLowInfo
41035.0	0.0	1.0	0.131	0.338	0.003	0.025	SendCreditLowInfo
0.0	0.0	0.0	0.0	0.0	0.0	0.0	GenerateBackOrde
0.0	0.0	0.0	0.0	0.0	0.0	0.0	GenerateBackOrde
0.0	0.0	0.0	0.0	0.0	0.0	0.0	ReleaseOrder (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	ReleaseOrder (occ)
30.0	400.0	400.0	400.0	0.0	0.0	0.0	path1 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path2 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path3 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path4 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path5 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path6 (dur)
140.0	400.0	400.0	400.0	0.0	0.0	0.0	path7 (dur)
50.0	400.0	400.0	400.0	0.0	0.0	0.0	path8 (dur)
40.0	400.0	400.000	400.000	0.0	0.0	0.0	path9 (dur)

Tabelle 1 - JSIM Statistik-Tabelle[1]

10: Num of Iterations			
	:BarnesGetPrice	:CheckCredit	:SendcreditLowInfo
	:1.913394	:1.342986	:0.759598
	:1.714697	:0.682717	:0.486978
	:1.562682	:1.42475	:0.692787
	:1.886854	:0.643781	:0.473613
	:1.828123	:0.816607	:0.525161
	:1.765729	:0.876123	:0.486001
	:1.502503	:0.93461	:0.534121
	:1.617242	:0.644926	:0.464647
	:2.485304	:0.82555	:0.691154
	:2.126191	:0.855064	:0.472463
	:1.84127	:0.918563	:0.57291
Total	:18.330595	:8.622691	:5.399835
Average	:1.8330595	:0.8622691	:0.5399835
Minimum	:1.502503	:0.643781	:0.464647
Maximum	:2.4853	:1.42475	:0.692787
STD	:0.276012	:0.2134661	:0.0824032

Tabelle 2 - Perl-Ausführung Statistik-Tabelle[1]

8 Zusammenfassung und Ausblick

Im Zusammenhang mit Web-Services wird die Dienstgüte ein wichtiger Faktor zur Bestimmung der Fähigkeiten eines Web-Services sein. Diese Arbeit hat Methoden zum Auswerten der Qualität der verschiedenen Kombinationen von Web-Services dargestellt. Dazu wurde der JSIM – Simulator und Perl-Execution-Code verwendet. Simulationen sind ein wichtiger Bestandteil bei der Auswertung eines Prozesses, da sie dem Entwickler erlauben, den Web-Prozess mit sehr geringen Kosten zu erforschen.
Die Weiterentwicklung von JSIM könnte folgende Punkte beinhalten[3]:
Die Fähigkeit zu haben, die Last auf den JSIM Facilities-Nodes dynamisch anpassen zu können.
Die Fähigkeit zum Monitoring der Delay-Message-Time in den Transportern (Links) zwischen JSIM-Facilitie-Nodes.

Literaturverzeichnis:

1- S. Chandrasekaran, J. A. Miller, G. S. Silver, B. Arpinar and A. P. Sheth.
Composition, Performance Analysis and Simulation of Web Services,
Department of Computer Science/LSDIS Lab. The University of Georgia
Athens, Georgia 30602-7404, U.S.A.

2- G. A. Silver, A. Maduko, R. Jafri, J. A. Miller, A. P. Sheth
Modeling and Simulation of Quality of Service for Composite Web Services,
Department of Computer Science, University of Georgia Athens, GA 30602, USA

3- John A. Miller, Yongfu Ge, Junxiu Tao.
Component-Based Simulation Environments, JSIM as a case study using java beans,
Computer Science Department 415 GSRC, University of Georgia , Athens, GA 30602-7404, U.S.A.

4-Anbazhagan Mani, Arun Nagarajan, Understanding Quality of Service for Web services,
(http://www106.ibm.com/developerworks/webservices/library/ws-quality.html)