Entwurf und Implementierung einer transaktionalen Subworkflowsteuerung

Diplomarbeit

Fakultät für Informations- und Kognitionswissenschaften,
Eberhard Karls Universität Tübingen

Thomas Hornung

Aufgabensteller:	Prof. Dr.-Ing. Wilhelm G. Spruth
Betreuer:	Dipl.-Inform. Joachim Franz
Abgabedatum:	25. September 2003

Erklärung der Selbständigkeit

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Kirchentellinsfurt, den 25. September 2003

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mir vor und während der Durchführung dieser Diplomarbeit geholfen haben und mich unterstützt haben.

Mein besonderer Dank gilt hier Herrn Joachim Franz, Prof. Dr. Wilhelm G. Spruth und Prof. Dr. Wolfgang Rosenstiel für die gute und kompetente Betreuung. Herrn Franz möchte ich danken für das starke persönliche Engagement und dafür, dass er sich stets Zeit nahm, um kritische Punkte zu diskutieren.

Darüber hinaus möchte ich Herrn Spruth und Herrn Franz für die Erlaubnis danken, für das Vorwort und die Motivation der Transaktionsmaschine wesentliche Teile aus den Fachaufsätzen [SP2] und [SP4] zu verwenden.

Danken möchte ich auch Herrn Marc Beyerle für die Unterstützung bei OS/390 und DB2 Fragen, seine Hilfe bei der Anpassung des in [BEY] beschriebenen C Launchers auf die Subworkflowsteuerung und die Erlaubnis den für die Performance-Messung relevanten Teil seines Source Codes für das ebenfalls in [BEY] beschriebene Basic Online-Banking System (BOBS), auf der begleitenden CD zu dieser Diplomarbeit aufführen zu dürfen.

Einige der IBM-Mitarbeiter möchte ich besonders hervorheben:

Herr Harald Bender für die Hilfe bei kritischen Fragen bei der RMF-Performance-Messung unter OS/390.

Und natürlich Herrn Uwe Denneler für seine Hilfe und persönliches Engagement bei der Realisierung der Performance-Messungen mit Hilfe des RMF-Performance-Monitor unter OS/390.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

In der klassischen Transaktionsverarbeitung wird das Problem von „long running transactions“ dadurch gelöst, dass man die Geschäftslogik in einzelne kleinere Units of Work (UoW) zerlegt, die dann in Form von „Short Running Transactions“ ausgeführt werden (siehe [BEN]).

Die in [SP2] beschriebene Transaktionsmaschine benutzt dieses Vorgehen, um einen Geschäftsprozess als Ganzes durch einen Masterflowkontroller (Masterflow Controller, kurz MFC) zu steuern, der in einzelne UoW, hier Subworkflows genannt, unterteilt wird. Diese werden dann von einer transaktionalen Subworkflowsteuerung (Subworkflow Controller, kurz SWFC) bearbeitet, wobei diese Subworkflows aus 1..n Aktivitäten bestehen, die 1..m ACID-Transaktionen (siehe [BER]) umfassen können (Abbildung 1). Aus diesem Grund spricht man hier auch von transaktionalen Workflows (siehe [LE1]).

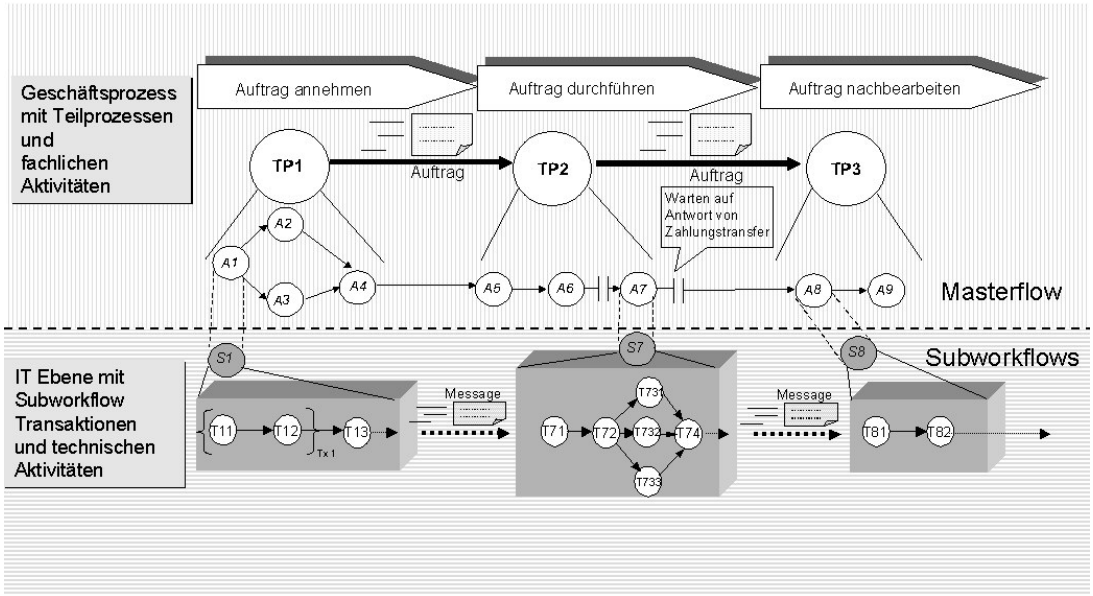

Abbildung 1: Geschäftsprozess mit seinen Teilprozessen und fachlichen Aktivitäten

Die Abbildung von fachlichen Aktivitäten auf die IT-Ebene erfolgt durch die Umsetzung auf Subworkflow Transaktionen mit ihren technischen Aktivitäten, wobei diese sowohl in Java als auch in COBOL entwickelt werden können. Dadurch ist sichergestellt, dass Legacy-Anwendungen, die großteils in COBOL entwickelt wurden, weiterhin benutzt und neue Anwendungen in Java entwickelt werden können.

Der Performance Einfluss dieser Subworkflowsteuerung wurde in der vorliegenden Diplomarbeit anhand einer beispielhaften Implementierung untersucht, wobei Performance-Messungen einen Performance-Vorteil von ca. 20,2 % für transaktionale Subworkflows gegenüber dem bisherigen Ansatz mit Einzeltransaktionen ergaben und zeigten, dass durch weitere Optimierungen eine absolute Steigerung der bisherigen Performance-Ergebnisse von ca. 85,2 % möglich ist.

Um die im transaktionalen Umfeld benötigten hohen Durchsatz-Anforderungen erfüllen zu können, wurde diese Subworkflowsteuerung (SWFC) nicht unter einer normalen Java Virtual Machine (JVM) implementiert, sondern in der von IBM entwickelten Persistent Reusable Java Virtual Machine (PRJVM) [JVM], die speziell für die Ausführung von Java Applikationen in Transaktionsumfeldern für OS/390 bzw. z/OS konzipiert wurde.

1 Einleitung

1.1 Begriffe

In dieser Diplomarbeit werden folgende Begriffe aus dem Bereich der Geschäftsprozessmodellierung verwendet:

- Geschäftsprozess, kurz Prozess: Ein Geschäftsprozess ist die Definition eines Ablaufes und den damit verbundenen Tätigkeiten, welcher zur Durchführung eines Geschäftes (Auftrags) notwendig ist. Ein typisches Beispiel eines Geschäftsprozesses ist die Beantragung eines Kredites,
- Workflow: Werden die einzelnen Tätigkeiten eines Geschäftsprozesses durch eine IT-Umgebung unterstützt, spricht man von Workflows (siehe [LE1]). Im Rahmen dieser Diplomarbeit werden sogenannte transaktionale Workflows behandelt. Dies sind Workflows, welche ACID (Atomicity, Consistency, Isolation, Durability)-Eigenschaften auf Workflow-Ebene garantieren (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows),
- Fachliche Aktivität: Die einzelnen Verarbeitungsschritte eines Geschäftsprozesses werden als fachliche Aktivitäten bezeichnet, z.B. Zahlungstransfer durchführen (siehe Abbildung 3),
- Technische Aktivität, kurz Aktivität: Die Realisierung der fachlichen Aktivitäten in einer IT-Umgebung werden als technische Aktivitäten bezeichnet, wobei hier eine fachliche Aktivität auch durch eine Transaktion, bestehend aus mehreren technischen Aktivitäten realisiert werden kann, z.B. eine Transaktion, bestehend aus zwei Java Methoden, welche eine Lastschrift und Gutschrift durchführen (siehe Abbildung 3).

Wenn es im Nachfolgenden nicht explizit differenziert wird, dann bezeichnet der Begriff Aktivität immer technische Aktivitäten.

1.2 Überblick

Die typischen Kernanwendungssysteme, die heute im Einsatz sind, wurden über mehrere Jahrzehnte entwickelt. Für diese bestehen nach Schätzungen 180 Milliarden Zeilen COBOL-Code, mit einer jährlichen Zuwachsrate von 5 Milliarden Zeilen [GA1], [GA2]. Nach [SPE] sind derzeit 20 Milliarden Zeilen CICS-Code in Benutzung. Darüber hinaus wird geschätzt, dass etwa 10 Millionen Mannjahre in die Entwicklung von unternehmenskritischen OS/390-Anwendungen unter CICS investiert wurden. Das bedeutet eine Investition von etwa einer Billion US-Dollar in OS/390-Anwendungssoftware unter

CICS [SPE]. Die Wartung und ständige Anpassung an sich ändernde Unternehmensbelange stellen einen erheblichen Kostenfaktor für die Unternehmen dar.

Umfragen bei großen Unternehmen (typischerweise S/390 mit z/OS) zeigen, dass die nächsten großen Herausforderungen, im Zeitraum bis 2005, der Umbau und das Reengineering der heutigen Kernanwendungssysteme sein wird und muss. Hierbei ist nicht mehr an eine Ablösung der Großrechnersysteme und deren Anwendungen gedacht. Vielmehr soll durch ein Reengineering eine Ausrichtung auf das Kerngeschäft der Unternehmen erfolgen. Es steht hierbei eine engere Ausrichtung der IT auf die Geschäftsstrategie der Unternehmen im Vordergrund [DA1], [DA2]. Dies bedeutet im Gegensatz zum teilweisen Reengineering (process improvement) eine grundlegende Neukonzeption, von den Geschäftsprozessen bis hin zu den IT-Systemen der Unternehmen.

Durch die aktuell erreichte technologische Spitzenposition der z/Series Server unter z/OS im Hinblick auf RAS (Reliability, Availability und Serviceability) und der hohen Verarbeitungsleistung (max. Durchsatz von Geschäftstransaktionen), eignen sich diese zentralen Server besonders gut als Zielplattform für weitere Serverkonsolidierungsaufgaben sowie als Basissystem für zukünftige Kernanwendungssysteme [SP1], [SP3].

In mehreren Projektstudien (Anwendungssysteme > 10 Millionen Zeilen COBOL-Code) haben sich eine Reihe von wesentlichen Anforderungen an eine zukünftige Anwendungs- und Systemarchitektur der neuen Kernanwendungssysteme herauskristallisiert. Diese wurden im Rahmen von Voruntersuchungen und Studien zu der Neuausrichtung der Kernanwendungssysteme bei Kunden im Bankenumfeld belegt und sind in Anforderungskatalogen konsolidiert.

Die folgende Liste zeigt exemplarisch einen Auszug:

- Mandantenfähigkeit (z.B. Mehrbankfähigkeit)

  Dies bedeutet im Wesentlichen die anwendungstechnische Unterstützung für eigenständige, juristisch unabhängige Unternehmen (z.B. eigenständige Bankinstitute mit eigenem Kundennetz, Kundenstamm, etc.). IT-Aspekte hierzu sind z.B. getrennte physische Datenhaltung; eine eventuelle Verschlüsselung von Transaktionsdaten, falls Aufträge von verschiedenen Mandanten auf dem gleichem System verarbeitet werden; Ablaufverfolgung der Aufträge und Abrechnung der erbrachten Leistungen pro Mandant; etc. Der Grad der Mandantenunterstützung (strikte oder schwache Mandantentrennung) wird durch eine gemeinsame Vereinbarung (Vertrag) mit dem Serviceanbieter (z.B. einer Transaktionsbank) geregelt. Diese Vereinbarung wird auch als „Service Level Agreement“ kurz SLA bezeichnet und enthält noch weitere durch den Mandanten zu bestimmende „Quality of Service“ Parameter. Anhand der festgelegten SLA Parameter bestimmt sich zum einen der Service (QoS), welcher für den Mandant erbracht wird, zum anderen der Preisrahmen des übergebenen Geschäftsauftrags, welcher bei der Abwicklung des Geschäftes berechnet wird.

- Mandanten und Kundeninformationen über ein Auftragsmanagement

  Dies erlaubt eine Sicht auf aktuelle Aufträge mit Status, Verarbeitungskosten und berechneter Fertigstellung. Die Mandaten und eventuell deren Kunden sollen jederzeit in der Lage sein, übergebene (Geschäfts-) Aufträge online nachverfolgen zu können und aktuelle Statusinformationen über den Stand der Bearbeitung, die aktuellen Kosten und dem voraussichtlichen Ende der Auftragsbearbeitung zu erhalten. Auch ein nachträgliches Beschleunigen oder Rückstellen (Um-Priorisierung) der Abarbeitung soll im Rahmen der vereinbarten Service Level Agreements „on the flight“ möglich sein. Storno und Abbruch des Auftrages kann auch durch den Mandanten jederzeit durchgeführt werden.

- Möglichkeit der Integration und Anbindung externer und interner Software Komponenten

  Hierunter zählen neben der Einbindung bzw. Integration von „zugekauften“ Softwarekomponenten auch die Anbindung von nachgelagerten Systeme wie z.B. Dokumentenarchivierung oder Finanzbuchhaltung. Auch externe Zuliefersysteme wie Börsensysteme, Mandantensysteme, oder Systeme der Bundesbank etc. müssen im Rahmen des Prozessablaufs angebunden werden.

- Hohe Flexibilität in der Einführung und Änderung neuer Produkte für die Kernanwendungssysteme mit mandantenspezifischen Verarbeitungspfaden

  Dies bedeutet, dass bei der Einführung und der Änderung von neuen Produkten des Serviceanbieters (z.B. Abwicklung von neuen Finanzprodukten im Rahmen der Riester-Rente, oder Anpassung von Produkten an Mandantenwünsche, europäische bzw. internationale Regeln) die Abbildung auf das Kernanwendungssystem so gestaltet sein soll, dass nur geänderte Abläufe zu einem bestehenden Basisprodukt eingebracht werden müssen, ansonsten die bestehenden Prozessabläufe aber mitverwendet werden können. Dies gilt auch bei Derivaten von Produkten mit geringer Abweichung im Prozessablauf. Die Einbringung und Änderung von Produkten sollen am laufenden Kernanwendungssystem möglich sein.

- Reduktion der Kosten pro Transaktion bei der Verarbeitung und Wartung

  Dieser Punkt spiegelt eines der Hauptprobleme der heutigen „organisch gewachsenen“ Kernanwendungssysteme wieder. Der Transaktionsdurchsatz soll (auch bei hohem Volumen > 1 Millionen Transaktionen pro Stunde) in Bezug auf benötigte Ressourcen nahezu linear skalieren, wobei der Durchsatz für das Standardgeschäft (Basisprodukt) durch das System autonom optimiert werden soll. Dies führt zu einer Minimierung der Transaktionskosten durch Erreichen des Skalierungseffektes (Economy of Scales).

- Enge Integration der Kernanwendungssysteme in das Systemmanagement

  Dies ist unter der Anforderung eines 24 Stunden / 7 Tage unterbrechungsfreien Betriebs absolut notwendig und bedeutet eine Einbindung in ein Monitoring (Überwa-

chung), Kapazitäts- / Performance-Management, Konfigurations- und Softwaremanagement, wie auch das Management von Wiederanlauf und der Ausfallsicherheit.

- Unterstützung und Umsetzung mehrerer externer Formate

  Bei Aufnahme und Übergabe von Geschäftsaufträgen an den Serviceanbieter durch Mandanten und externe Systeme sind eine Reihe von verschiedenen Eingangs- und Ausgangsformaten, Kommunikationswegen und Protokollen zu unterstützen. Diese sind durch das Kernanwendungssystem neutral zu behandeln damit eine einheitliche, format-unabhängige Verarbeitung durchgeführt werden kann.

Die grobe Einschätzung der Relevanz für die zukünftige Anwendungs- bzw. Systemarchitektur enthält bereits das Paradigma „strikte Trennung der Geschäfts- bzw. Fachlogik von der technischen Logik und Steuerung“. Als dominante Eigenschaften einer zukünftigen Systemarchitektur wurden die Anforderungen Performance, Echtzeitfähigkeit, unterbrechungsfreier Betrieb, Restart-Fähigkeit, komponentenbasiertes System, Prozesssteuerung und Multiplattformfähigkeiten ermittelt.

1.3 Komponenten der Transaktionsmaschine

Die meisten der in Kapitel 1.2: Überblick beschriebenen Anforderungen lassen sich nur schwer mit den bestehenden Kernanwendungssystemen implementieren. Verschiedene Machbarkeitsstudien haben gezeigt, dass dies nicht ohne ein grundlegendes Re-Design der Anwendungssysteme möglich ist [DA2], [DA3]. Der Grund hierfür liegt hauptsächlich darin, dass ein monolithisches Design mit redundanter Funktionalität in Anwendungsverarbeitung, Formaten und Datenstrukturen zu einem nicht-skalierbaren, zufälligen Anwendungs-Workload führt. Ein geschäftsprozessorientierter Ansatz in der Umsetzung von Geschäftsprozessen auf die entsprechende IT-Infrastruktur hat sich als erfolgversprechend gezeigt [HAM], [MAR], [GRO], [DAV].

Die in Abbildung 4 gezeigte Systemarchitektur der Transaktionsmaschine setzt zum einen diese Anforderungen um, zum anderen bildet sie ein neues Verarbeitungskonzept ab. Dies geschieht mit dem Ziel, Geschäftsprozesse als Ganzes zu steuern und optimal unter Nutzung einer transaktionalen Verarbeitung (mit ACID Eigenschaften) auszuführen. Der gewählte Begriff „Transaktionsmaschine" erweitert dabei das Konzept des Transaktionsmonitors (Transaction Processing Monitor) zu einem generellen Ansatz [GRA]. Es handelt sich hierbei um eine transaktionsorientierte Prozesssteuerung, welche Aufträge (Service Request Work Unit, kurz SRWU) entgegennimmt und diese autonom, an vereinbarten Serviceanforderungen (Service Level Agreement, kurz SLA) orientiert, flexibel steuert und abarbeitet. Das Ergebnis der Verarbeitung sind abgearbeitete Aufträge (Ergebnisaufträge).

Die Transaktionsmaschine besteht aus folgenden Basiskomponenten:

- Kanalkontroller (Channel Controller) und Profiler

Nimmt eingehende Aufträge von verschiedenen Kanälen (z.B. Internet, Filetransfer, Datenträger, Remote Job Entry, Channel-to-Channel, etc.) entgegen und versendet ausgehende Ergebnisse (Ergebnisaufträge, Responses) bzw. Anforderungen (externe Systemanfragen) über die verschiedenen Kanäle. Der Kanalkontroller formatiert alle eingehenden Aufträge in ein „neutrales“ Verarbeitungsformat (SRWU), um eine einheitliche Verarbeitung zu gewährleisten. In der Ausgaberichtung werden die verschiedenen Ausgabeformate erzeugt. Man spricht hier auch von einem „Multi-Kanalkontroller“. Die Profiler Komponente des Kanalkontrollers ermittelt die Verarbeitungsdomain (Cluster von Verarbeitungsknoten), welche unter den geforderten Rahmenbedingungen, wie Service Level Vereinbarungen, Auftragstyp, Priorität, aktuelle Systemkapazität, den benötigten Verarbeitungseinheiten (Service Units), etc. den Auftrag abarbeiten kann.

Er führt abhängig von den vereinbarten Serviceanforderungen eine adaptive service-orientierte Lastverteilung zwischen Verarbeitungsdomainen mit der gleichen QoS (Quality of Service) Charakteristik durch.

- Requestmanager

  Verwaltet die verschiedenen Warteschlangen und Ereignislisten der Transaktionsmaschine, über welche die verschiedenen Komponenten kommunizieren.

- Masterflowkontroller (Masterflow Controller, kurz MFC)

  Verteilt die Aufträge innerhalb einer Verarbeitungsdomain (Cluster von Verarbeitungsknoten) auf einen geeigneten Carrier. Die Attribute eines Carriers werden auf der Ebene der Masterflowdefinitionen festgelegt. Der Masterflowkontroller beinhaltet die Grobsteuerung der Prozesse bis auf die Ebene der fachlichen Aktivitäten. Er bildet den service-orientierten Auftrag an die Verarbeitungsdomain auf eine lokale Workloadmanager-Sicht ab (ressourcen-orientiert).

- Carrier

  Ist ein Container (Trägersystem), welcher eine gesicherte Transaktionsumgebung für die Ausführung der Geschäftslogik (Aktivitäten) bereitstellt. Er bildet quasi die „äußere“ Hülle für eine fachliche Aktivität des Masterflowkontrollers, welche auch die globalen und lokalen Daten (Nachrichten), die zwischen den Aktivitäten ausgetauscht werden, mit verwaltet.

- Subworkflowsteuerung (Subworkflow Controller, kurz SWFC)

  Führt innerhalb des Carriers (Containers) die technischen Transaktionen auf Workflow-Ebene aus und stellt die transaktionalen Eigenschaften der Workflows sicher (short running). Hierbei ist die Abbildungsvorschrift so, dass eine fachliche Aktivität des Masterflowkontrollers auf eine Subworkflowkette passt. Eine Subworkflowkette besteht aus 1..n technischen Aktivitäten.

  Der gesamte Subworkflow kann darüber hinaus 1..m technische Transaktionen umfassen (logical units of work) (siehe Abbildung 2 und Abbildung 3). Diese Transak-

tionen müssen ACID (Atomicity, Consistency, Isolation, Durability) Eigenschaften besitzen, welche durch Services des Carriers (z.B. Encina, CICS, IMS, Tuxedo) mit unterstützt werden [LE1].

- Fachkomponenten

  Die Anwendungslogik (Abbildung 4, Geschäftslogik) fügt sich in Form von einfachen, flexibel konfigurierbaren technischen Aktivitäten (Subworkflowaktivitäten) ein. Dadurch, dass die Subworkflowsteuerung sowohl Java als auch COBOL Aktivitäten unterstützt, können hierbei auch in COBOL geschriebene, bestehende Legacy Systeme eingebunden werden. Eine bestimmte Fachkomponente kann aus einer oder mehreren Subworkflowketten bestehen, welche jeweils eine nicht unterbrechbare Abarbeitung (auf der Flow-Ebene) von technischen Aktivitäten darstellen. Die Granularität der einzelnen technischen Aktivitäten ist hierbei entscheidend für den Durchsatz und die Komplexität in der Verwaltung des Anwendungssystems. Ein zu grober Zuschnitt schränkt die Flexibilität bei den Kombinationsmöglichkeiten von technischen Aktivitäten und die notwendige Unterbrechbarkeit stark ein (long running transaction Problem [BEN]). Ein zu feiner Zuschnitt stellt ein Mengenproblem der Verwaltung von technischen Aktivitäten in der Breite dar und produziert in Summe sehr lange Subworkflowketten mit entsprechend höherem „Dispatching“ Overhead seitens der Transaktionsmaschine. Außerdem kann hierbei der Bezug zu den fachlichen Aktivitäten der Geschäftsprozessebene (Masterflowkontroller) verloren gehen, wenn die technische Steuerungsebene zu fein-granular arbeitet.

- Systemkomponenten

  Die Transaktionsmaschine ist über eine Plug-In Schnittstelle durch Systemkomponenten erweiterbar, welche nicht in der Basis voll ausgeprägt sind (Abbildung 4, Systemkomponenten Plug-Ins). Diese Schnittstelle bedient die operative Ablaufverfolgung (Monitoring), die fachliche Ablaufverfolgung (Tracking), die Verwaltung von Benutzern, Rollen, Rechte und Ablaufdefinitionen (Administration) sowie einen Auftragsinformationsdienst (Informationsservice).

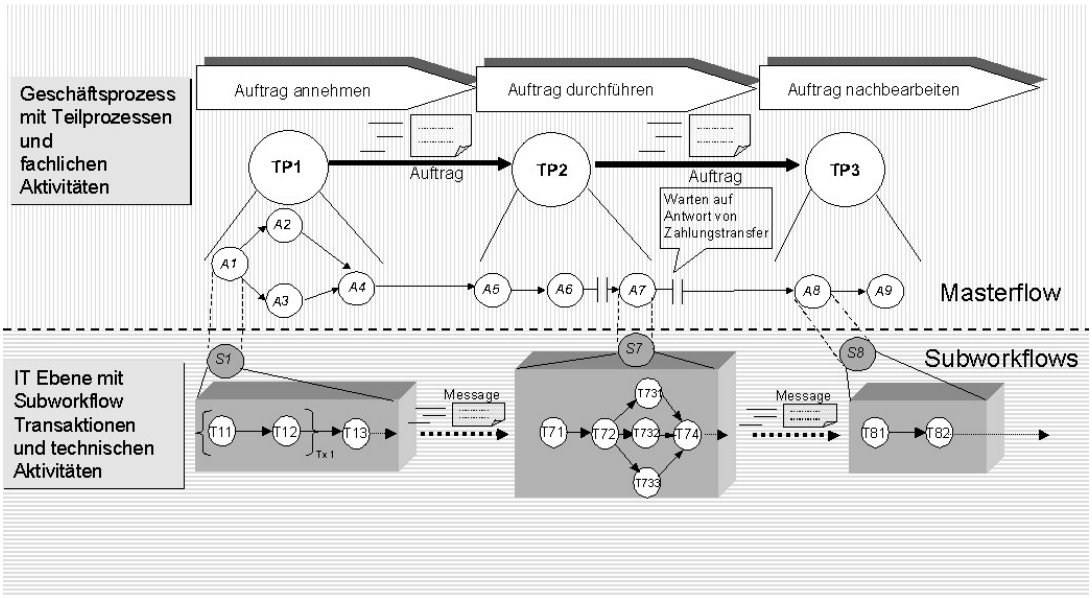

Abbildung 2: Geschäftsprozess mit seinen Teilprozessen und fachlichen Aktivitäten

Abbildung 3: Ausschnitt der fachlichen und technischen Aktivitäten von Abbildung 2

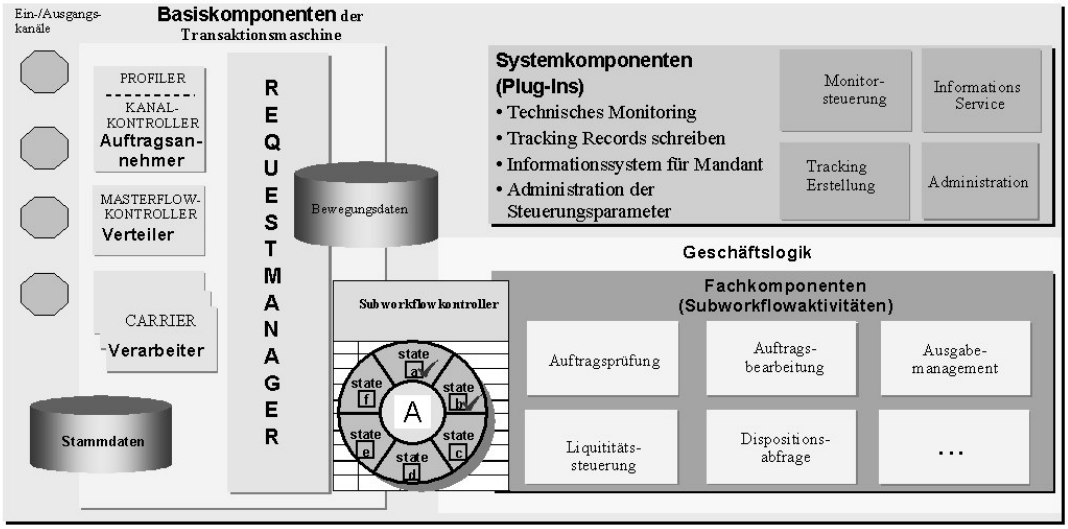

Abbildung 4: Gesamtdarstellung der Transaktionsmaschine und der Fachkomponenten

Das in Abbildung 5 dargestellte Szenario zeigt eine 3-Tier Anwendungsarchitektur auf Basis der Transaktionsmaschine in einer IMS Umgebung (ein ähnliches Szenario wäre auch für CICS denkbar). Hierbei läuft ein Teil der Transaktionsmaschine auf dem Middle-Tier Server und ein Teil auf dem Backend-Server unter IMS als Trägersystem (Carrier), welches als Transaktionsmonitor bei der Abarbeitung des Subworkflows (technische Aktivitäten) genutzt wird.

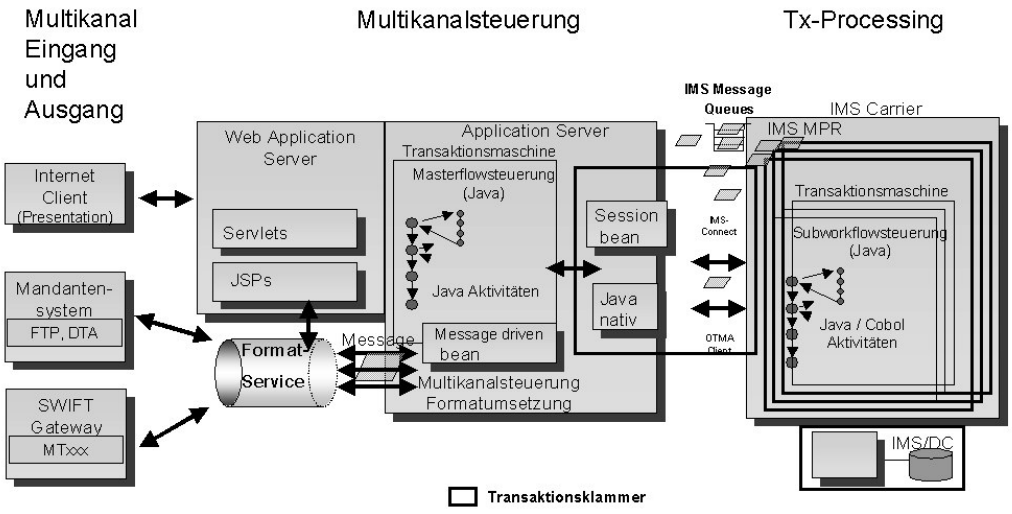

Abbildung 5: Architekturszenario auf Basis einer 3-Tier Anwendungsarchitektur mit IMS Subworkflowsteuerung

1.4 Aufgabenstellung und Ziele der Diplomarbeit

Die Aufgabenstellung dieser Arbeit ist die vollständige Spezifikation der Funktionalität der Subworkflowsteuerungs-Komponente der Transaktionsmaschine anhand der in [JAC] beschriebenen Use Cases und der Entwurf einer Beschreibungssprache für die Subworkflows auf Basis der in [CUR] definierten Business Process Execution Language for Web Services, welche die in Abbildung 6 dargestellten Workflow Schemata unterstützt.

Darüber hinaus wird anhand dieser Spezifikation ein Prototyp entworfen und in großen Teilen implementiert, der den in der Diplomarbeit von Marc Beyerle (siehe [BEY]) implementierten Persistent Reusable Java Virtual Machine (PRJVM) Launcher als Trägersystem (Carrier) nutzt und die Ausführung von transaktionalen Subworkflows ermöglicht.

Abschließend werden anhand von Performance Messungen Flaschenhälse (Bottlenecks) in der bestehenden Implementierung des Prototypen untersucht und Verbesserungen vorgeschlagen.

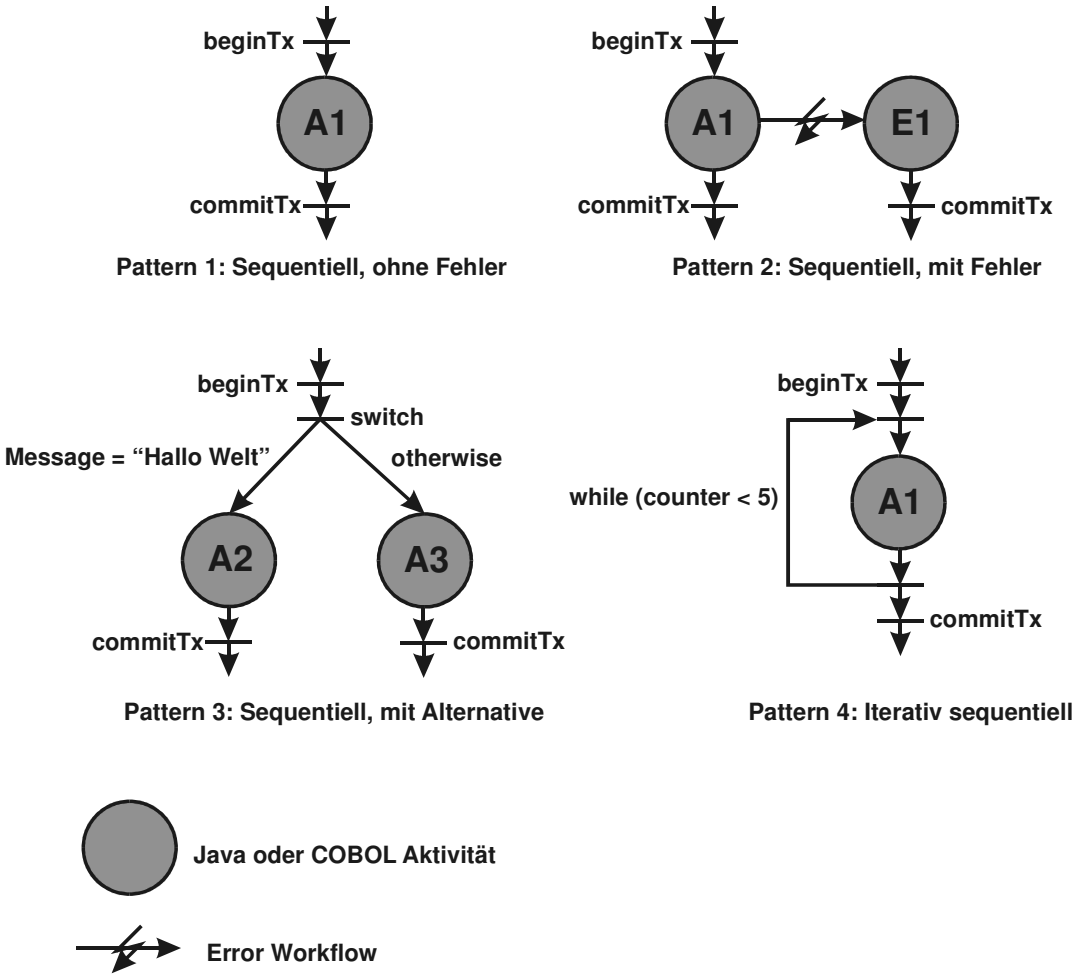

Abbildung 6: Workflow Schemata

1.5 Struktur der Diplomarbeit

1.5.1 Strukturierung der Kapitel

- Kapitel 2 gibt einen generellen Überblick über die Funktionalität der Subworkflowsteuerung und beschreibt die verschiedenen Komponenten der Subworkflowsteuerung.
- Kapitel 3 beschreibt für die Erstellung eines Subworkflows notwendige Schritte.
- Kapitel 4 beschreibt die Ausführung eines Subworkflows.
- Kapitel 5 beschreibt den Aufruf von COBOL Aktivitäten.
- Kapitel 6 beschreibt die Erweiterung der Systemfunktionalität mit Hilfe von Plug-In Aktivitäten.
- Kapitel 7 beschreibt die Fehlerbehandlung des Subworkflows.

- Kapitel 8 beschreibt die transaktionale Steuerung des Subworkflows.
- Kapitel 9 diskutiert die Ergebnisse der durchgeführten Performance-Tests.
- Kapitel 10 fasst die Ergebnisse der Diplomarbeit zusammen und gibt einen Ausblick auf zukünftige Erweiterungen und mögliche Fortführungen dieser Diplomarbeit.
- Kapitel 11 enthält das Abkürzungsverzeichnis.
- Kapitel 12 enthält das Literaturverzeichnis.
- Kapitel 13 enthält den Anhang, der folgende Unterkapitel umfasst:
  - Kapitel 13.1 listet den Inhalt der beigelegten CD auf,
  - Kapitel 13.2 enthält die Spezifikation der Subworkflow Definition Language (SWFDL), die es ermöglicht, Subworkflows für die Subworkflowsteuerung zu definieren,
  - Kapitel 13.3 beschreibt exemplarisch einen Subworkflow mit dazugehöriger WSDL-Schnittstelle und SWFDL-Beschreibung,
  - Kapitel 13.4 beschreibt die Ausführung eines Subworkflows mittels einer Finite State Machine (FSM),
  - Kapitel 13.5 enthält eine Liste der möglichen Return Codes,
  - Kapitel 13.6 enthält die Use Cases (siehe [JAC]), welche die Funktionalität der Subworkflowsteuerung definieren,
  - Kapitel 13.7 beschreibt die Konfiguration des Testsystems, welches für die Performance Messungen eingesetzt wurde und die verschiedenen gemessenen Szenarien mit Ergebnissen,
  - Kapitel 13.8 enthält eine Beispiel-Konfigurationsdatei für die Subworkflowsteuerung,

Kapitel 13.9 enthält eine Beispiel-Konfigurationsdatei für die Persistent Reusable Java Virtual Machine (PRJVM).

1.5.2 Anmerkung

An verschiedenen Stellen in dieser Ausarbeitung finden sich *Anmerkungen zur Implementierung*, die den Umfang der Implementierung der entsprechenden Funktionalität in der vorliegenden Fassung dokumentieren und *Siehe Use Case*-Referenzen, die auf den entsprechende Use Case zu dieser Funktionalität verweisen.

Davon abgesehen ist dieser Diplomarbeit eine CD beigelegt, welche lokale Kopien der Internet Referenzen, ausgewählte referenzierte Publikationen, den Source Code der Subworkflowsteuerung, eine Online-Version der Java Dokumentation und diese Ausar-

beitung im pdf-Format enthält. Alternativ kann der Inhalt dieser CD unter folgender Internetadresse abgerufen werden:

http://www-ti.informatik.uni-tuebingen.de/~csp/diplom/thornung

2 Komponenten der Subworkflowsteuerung

2.1 Überblick

Die Subworkflowsteuerung ist zuständig für die transaktionale Verarbeitung der Masterflow Aktivitäten (siehe Abbildung 1). Dazu benutzt sie ein Carrier System (z.B. ein Transaktionsmonitor, Java Virtual Machine, Batch) als Laufzeitumgebung (runtime environment), welches Dienste für die Umsetzung der ACID Eigenschaften der Subworkflow Transaktionen zur Verfügung stellt [GRA]. Die Subworkflowsteuerung folgt dem Design Pattern eines transaktionalen Flowkontrollers mit transaktionalen Primitiven [LE1].

Darüber hinaus unterstützt die Subworkflowsteuerung eine Fehlerbehandlung über alternative Error Workflows [CHE] und bietet ein Programmiermodell (Application Programming Interface, kurz API, siehe Kapitel 3.2: Programmiermodell für Java und COBOL Aktivitäten) für die Aktivitäten, welches es ermöglicht:

- Nachrichten an den Masterflowkontroller bzw. den „Auftraggeber“ zu senden (siehe Kapitel 4.5: Senden von Nachrichten an den MFC (Notification API)),
- Über eine lokale Sicht auf die entsprechenden Datenschnittstelle transparent auf die für diese Aktivität benötigten Daten zuzugreifen,
- Den fachlichen Ablauf der Verarbeitung zu verfolgen (Tracking, realisiert als Plugin-Aktivität),
- Java und COBOL Aktivitäten auszuführen,
- Der Subworkflowsteuerung zu signalisieren, Aktivitäten erneut zu starten bzw. einen Rollback oder das Zurücksetzen auf den letzten Savepoint (Restore) durchzuführen.

2.2 Persistent Reusable Java Virtual Machine (PRJVM)

Für die Ausführung von Transaktionen müssen die ACID (Atomicity, Consistency, Isolation, Durability)-Eigenschaften sichergestellt werden. Besonders die Isolationseigenschaft erfordert es aber, dass die Java Virtual Machine (JVM) nach jeder Ausführung einer Transaktion neu gestartet wird, damit die nachfolgende Transaktion nicht durch Änderungen der vorherigen beeinflusst wird.

Da dies ein sehr zeitaufwendiger Vorgang ist, wurde von IBM die Persistent Reusable Java Virtual Machine (PRJVM, siehe [JVM] und [BEY] entwickelt, die für die Ausführung von Java Applikationen in Transaktionsumfeldern für OS/390 bzw. z/OS konzipiert wurde. Diese ermöglicht es die JVM nach jeder Transaktion auf einen „sauberen“ Zu-

stand zurückzusetzen (reset), wenn gewisse Regeln eingehalten werden. Ermöglicht wird dies dadurch, dass man Java Klassen in zwei verschiedene Kategorien einteilt: in „vertrauenswürdige“ (trusted) Middleware Klassen, die „unbeschränkte“ Ausführungsrechte besitzen, und Applikationsklassen, die „beschränkte“ Ausführungsrechte besitzen.

Insbesondere gilt, dass zum Zeitpunkt des Zurücksetzens auf einen „sauberen“ Systemzustand keine Middleware Klasse eine Referenz auf eine Applikationsklasse halten darf.

Anmerkung zur Implementierung: Die Subworkflowsteuerung wurde so entwickelt, dass die Systemklassen der Steuerung als vertrauenswürdige Middleware Klassen implementiert wurden und die eigentlichen Transaktionsklassen, welche die Geschäftslogik bzw. die Subworkflows repräsentieren, als Applikationsklassen.

2.3 Service Request Work Unit (SRWU)

Die Service Request Work Unit beschreibt einen Auftrag für die Transaktionsmaschine. Sie ist unterteilt in verschiedene Segmente, die Kontroll- und Verarbeitungsdaten für die jeweilige Komponente der Transaktionsmaschine enthalten. Der Teil dieser SRWU, welcher die relevanten Daten für die Subworkflowsteuerung enthält, wird im Nachfolgenden Subworkflow Controller Request Work Unit (SWFC RWU) genannt. Dieses Segment enthält die Daten, welche für die Ausführung der Aktivitäten benötigt und von einem Message Container verwaltet werden.

Die Daten, welche dieser Message Container verwaltet, lassen sich in zwei Kategorien unterteilen: globale und lokale Daten: globale Daten haben eine Gültigkeit zwischen den Masterflow Aktivitäten (d.h. zwischen mehreren Subworkflows), wohingegen lokale Daten nur zwischen den SWF Aktivitäten bzw. innerhalb von SWF Transaktionen Gültigkeit haben. Der Unterschied ist, dass globale Datenelemente über Transaktionsgrenzen bzw. SWF Grenzen hinaus bestehen bleiben, während die lokalen Datenelemente nur innerhalb eines SWF (kann mehrere Transaktionen umfassen) bestehen, d.h. temporär sind.

Das SWFC Pendant der SWFC RWU, welche den Auftrag intern repräsentiert, ist die Java RWU. Sie wird zu Beginn vom I / O Controller aus der SWFC RWU erzeugt (siehe Kapitel 4.3: Initialisierung eines Subworkflows).

2.4 Message Container

Der Message Container verwaltet die globalen und lokalen Daten und bietet folgende Funktionalität nach außen:

- Mengenorientiertes Datenmanipulation API auf einer lokalen Sicht des Message Containers für die Aktivitäten. Dieses API ermöglicht insbesondere das mengenori-

entierte Aufsuchen und die mengenorientierte Aktualisierung von globalen und lokalen Daten (siehe Kapitel 4.4: Verwaltung der globalen und lokalen Daten).

- Ausgabe der globalen Daten als Java RWU Objekt,
- Ausgabe der für eine COBOL Aktivität benötigten globalen und lokalen Daten als Java COBOL Objekt über das COBOL API bzw. Synchronisation des (veränderten) Java COBOL Objektes mit den globalen und lokalen Daten im Message Container über das COBOL API nach dem COBOL Aufruf (siehe Kapitel 5: Aufruf von COBOL Aktivitäten). Dieses Java COBOL Objekt ist das Pendant der lokalen Sicht auf den Message Container, welche für Java Aktivitäten zur Verfügung gestellt wird,
- Rückschreiben von geänderten globalen und lokalen Daten (bei commitTx) inklusive zusätzlicher Statusinformationen in entsprechende Datenbanken bzw. Dateien mit Hilfe des Data Service bzw. Rücksetzen der Daten auf den Stand beim letzten beginTx bzw. commitTx. Das Rücksetzen der Daten auf den Stand beim letzten beginTx bzw. commitTx liefert als Resultat die Statusinformationen für den SWF Interpreter,
- Persistierung der globalen und lokalen Daten inklusive zusätzlicher Statusinformationen für den SWF Interpreter an Savepoints, bzw. Rücksetzen der Daten auf beliebige vorherige Savepoints mit Hilfe des Data Service. Das Rücksetzen der Daten auf vorherige Savepoints liefert als Resultat die Statusinformationen für den SWF Interpreter.

2.5 Finite State Machine

Die Ausführung eines Subworkflows wird mit Hilfe einer Finite State Machine (FSM) modelliert (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM)). Hierbei entsprechen die Zustände der FSM verschiedenen Stadien der Ausführung des Subworkflows – z.B. ist der Zustand Initiated SWF Activity bei der Ausführung einer Java Aktivität dadurch gekennzeichnet, dass eine lokale Sicht des Message Containers erstellt wurde (siehe Kapitel 4.4: Verwaltung der globalen und lokalen Daten) – und Zustandsübergänge entsprechen Verarbeitungsschritten, welche einen Übergang von einem Stadium in der Ausführung des Subworkflows in ein anderes bewirken – z.B. wird in der Execute SWF Activity-Phase die aktuelle Aktivität ausgeführt, was mit einem Zustandswechsel von Initiated SWF Activity nach Executing SWF Activity verbunden ist. Mögliche Zustandsübergänge werden als Ereignisse an den SWFC „gesendet“ (z.B. über einen Funktionsaufruf), der dann abhängig vom aktuellen Zustand bewerten kann, ob das ankommende Ereignis in diesem Zustand legal ist und der damit verbundene Verarbeitungsschritt ausgeführt werden soll oder nicht.

2.6 Subworkflow Definition Language (SWFDL)

Die SWFDL ermöglicht es Subworkflows zu spezifizieren und ist ein Subset der in [CUR] definierten Business Process Execution Language for Web Services (BPEL4WS), wobei SWFDL die Möglichkeiten zu Beschreibung des Kontrollflusses, die BPEL4WS bietet, nutzt und die Sprache um Verben für die Beschreibung des transaktionalen Kontext erweitert (siehe Kapitel 13.2: Subworkflow Definition Language (SWFDL)).

2.7 Plug-In Aktivitäten

Die Subworkflowsteuerung ermöglicht es nachträglich Systemfunktionalität, wie z.B. den fachlichen Ablauf der Verarbeitung verfolgen zu können, über sogenannte Plug-In Aktivitäten hinzuzufügen. Diese Plug-Ins können wie „normale“ (technische) Aktivitäten sowohl in COBOL als auch in Java implementiert werden und sind in Kapitel 6: Plug-In Aktivitäten beschrieben.

2.8 Module der Subworkflowsteuerung

Um ein komponentenbasiertes System zu realisieren wurde der SWFC in verschiedene Module unterteilt (Abbildung 7):

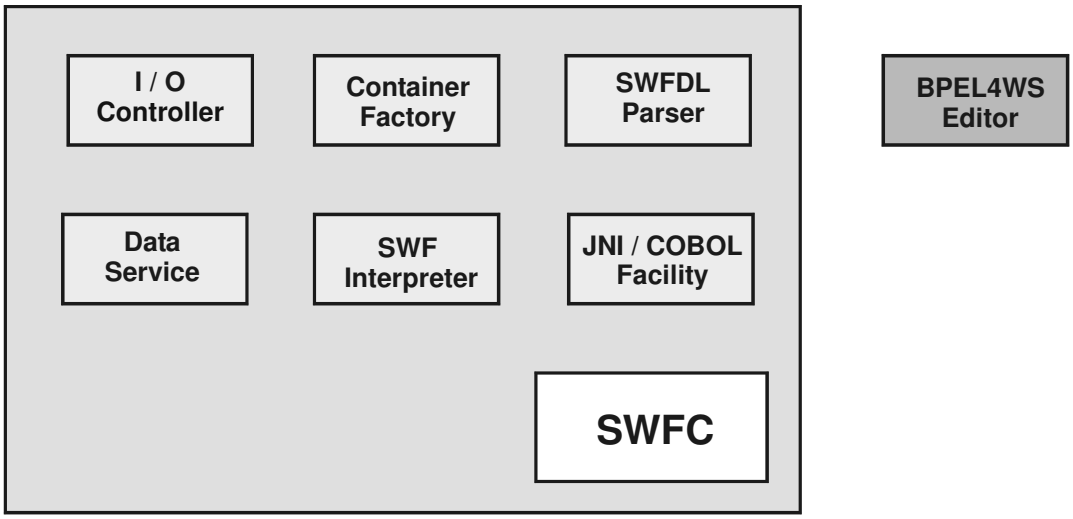

Abbildung 7: Überblick der SWFC Module

- Für jeden SWF wird mit Hilfe eines BPEL4WS-Editors eine Subworkflow Definition Language (SWFDL)-Beschreibung (siehe Kapitel 13.2: Subworkflow Definition Language (SWFDL)) generiert,
- Der SWFC regelt das Zusammenspiel der einzelnen Module und realisiert die Finite State Machine (FSM) des Subworkflows (siehe Kapitel 13.4: Beschreibung der

Finite State Machine (FSM)). Um die Administration des SWFC zu erleichtern, können verschiedene Parameter über eine Konfigurationsdatei (Property File, siehe Kapitel 13.8: Beispiel-Konfigurationsdatei für die Subworkflowsteuerung) eingestellt werden.

- Der I / O Controller ist zuständig für das Entgegennehmen von Aufträgen (SWFC Request Work Units – SWFC RWUs) vom MFC bzw. für die Rückgabe von bearbeiteten Ergebnisaufträgen (Ergebnis SWFC RWUs) an den MFC. Die Kommunikation zwischen MFC und SWFC erfolgt dabei über Ein- und Ausgabe-Warteschlangen (In und Out Queues), die der Carrier zur Verfügung stellt, unter dem der SWFC läuft. Alternativ kann der SWFC Aufträge direkt über eine Web Service Schnittstelle annehmen (siehe Abbildung 10).

  Zusätzlich bietet der I / O Controller ein Notification API, um Nachrichten an den MFC zu „senden“ (um z.B. Fehler oder die erfolgreiche Bearbeitung einer Aktivität an den MFC zu signalisieren, siehe Kapitel 4.5: Senden von Nachrichten an den MFC (Notification API)),

- Die Container Factory erzeugt aus der SWFC RWU einen Message Container (siehe Kapitel 4.3: Initialisierung eines Subworkflows, der die globalen und lokalen Daten verwaltet und ein mengenorientiertes API zur Verfügung stellt (siehe Kapitel 4.4: Verwaltung der globalen und lokalen Daten),

- Der SWFDL Parser erzeugt aus der SWFDL-Beschreibung eine SWF Repräsentation (z.B. ein Java Objektmodell, welches den entsprechenden Syntax Graphen repräsentiert, siehe [WIR] und [TUC]),

- Der Data Service bietet eine Schnittstelle zum Abfragen und Schreiben von Daten aus bzw. in Datenbanken und Dateien. Hierbei soll eine Optimierung gegenüber einem direkten Zugriff auf die entsprechende Datenbank durch einen effizienten Algorithmus erreicht werden, der die bereits gelesen Daten bzw. die geänderten Daten auf einem hoch-performanten Datenträger zwischenspeichert, um die Anzahl der realen Datenbankzugriffe zu minimieren,

- Der SWF Interpreter führt die SWF Repräsentation (und somit die Aktivitäten) aus und liefert als Resultat eine Ergebnis SWFC RWU,

- Die COBOL / JNI Facility realisiert den Aufruf von COBOL Aktivitäten.

Anmerkung zur Implementierung: Der im Rahmen dieser Diplomarbeit implementierte Data Service ermöglicht nur den Zugriff auf eine fixe Datenbank, und enthält keinen Algorithmus für die Zwischenspeicherung von Daten auf hoch-performante Datenträger.

3 Erstellung eines Subworkflows

3.1 Überblick

Für jeden Subworkflow (SWF) wird mit Hilfe eines Tools ein logisches Datenmodell erstellt, welches die globalen und lokalen Daten eines SWF beschreibt. Aus diesem logischen Datenmodell werden anschließend automatisch folgende Strukturen generiert:

- Eine Data Section Copystruktur für die SWFC RWU, welche die Reihenfolge der Felder in der SWFC RWU und deren Name und Typ spezifiziert. Diese wird vom I / O Controller verwendet, um das Java RWU Objekt zu erzeugen,
- Die anwendungs- bzw. kundenspezifischen Tabellen für die 1..n Datenbanken, die für die Datensicherung bei commitTxs durch die Data Service Schnittstelle benötigt werden,
- Die für die Data Service Schnittstelle benötigten Mappingstrukturen (hier Data Description genannt), um die globalen und lokalen Daten zu Beginn (sofern nötig) aus 1..n Datenbanken zu initialisieren bzw. bei Änderungen wieder rückzuschreiben.

Je nach gewählter Implementierung kann es sinnvoll sein, zusätzlich folgende Strukturen zu generieren:

- Eine Datenbanktabelle, welche der Struktur der lokalen und globalen Daten des logischen Datenmodells entspricht, die der Sicherung der Daten bei Savepoints dient,
- Zu jedem komplexen Datentyp eine entsprechende Java Datenklasse bzw. eine Java Klasse, welche die gesamte SWFC RWU repräsentiert.

Anschließend wird mit Hilfe eines BPEL4WS Editors der eigentliche Workflow als SWFDL-Datei erstellt und die Java bzw. COBOL-Aktivitäten analog zu dem in Kapitel 3.2 beschriebenen Programmiermodell für Java und COBOL Aktivitäten implementiert.

Anmerkung zur Implementierung: Die Persistent Reusable Java Virtual Machine [JVM] ermöglicht es mehrere Java Virtual Machines (JVM) als ein JVM Set zu starten, so dass mehrere Virtual Machine Instanzen parallel ausgeführt werden. In diesem Umfeld ist es wichtig, dass eine SWFDL-Datei pro JVM verfügbar ist. Davon abgesehen, müssen die oben beschriebenen Strukturen in der vorliegenden Implementierung manuell erstellt werden und die Namen der globalen und lokalen Daten in der SWFDL-Datei müssen eindeutig sein.

3.2 Programmiermodell für Java und COBOL Aktivitäten

3.2.1 Java Aktivitäten

Jede Java Aktivität bekommt beim Aufruf eine lokale Sicht auf den Message Container übergeben, welche die entsprechenden globalen und lokalen Daten verwaltet, die in der SWFDL Input und Output Message spezifiziert wurden. Um Daten abzufragen, zu ändern, etc. bietet die lokale Sicht ein Datenmanipulation API, welches das mengenorientierte Aufsuchen und die mengenorientierte Aktualisierung von globalen und lokalen Daten ermöglicht. (siehe Kapitel 4.4: Verwaltung der globalen und lokalen Daten und Abbildung 8). Außerdem kann eine Aktivität über das Notification API eine Nachricht an den MFC senden (siehe Kapitel 4.5: Senden von Nachrichten an den MFC (Notification API) und Abbildung 8).

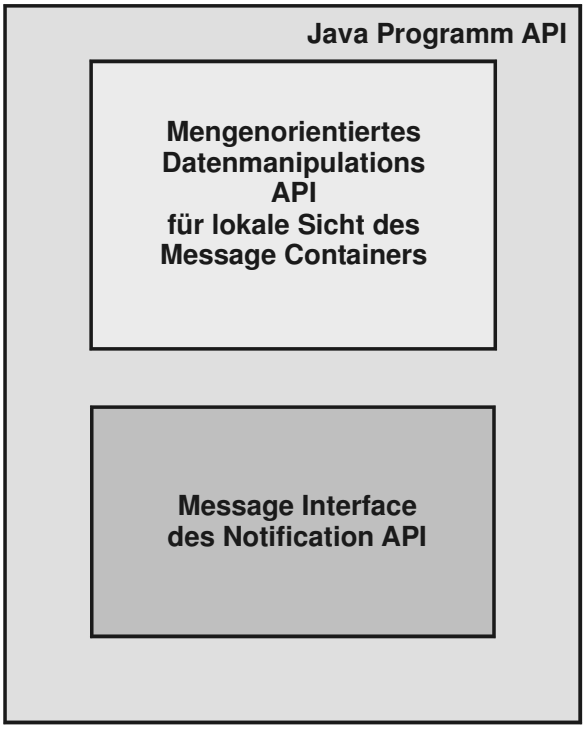

Abbildung 8: Programm API-Übersicht für Java Aktivitäten

Über den Fehler Return Code (siehe Kapitel 13.5: Liste der Return Codes) kann die Aktivität Fehler in der Ausführung der Aktivität an den SWF Interpreter signalisieren und somit die Ausführung des SWF direkt beeinflussen; so kann die Aktivität dem SWF Interpreter z.B. anzeigen, dass bei der Ausführung der Aktivität ein schwerwiegender Fehler aufgetreten ist, der es erfordert auf den letzten Savepoint zurückzusetzen (RC_RESTORE).

Anmerkung zur Implementierung: Generell gilt, dass es pro SWF nur eine Java Klasse geben darf, deren Methoden die Java Aktivitäten des SWF implementieren. Davon

abgesehen wird nur ein einfaches Zugriffs API unterstützt, das es ermöglicht globale und lokale Daten per Namen abzufragen.

3.2.2 COBOL Aktivitäten

Jede COBOL Aktivität bekommt beim Aufruf einen gefüllten Datenpuffer übergeben, der folgendermaßen strukturiert ist: zuerst kommen die Eingabedaten, welche in der Input Message spezifiziert wurden, dann die Ausgabedaten, die in der Output Message spezifiziert wurden (sofern vorhanden) und zum Schluss der Fehler Return Code, in dem die Aktivität eventuelle Fehler nach außen signalisieren kann. Dieser Datenpuffer wird von der COBOL Aktivität gegebenenfalls verändert und dient als Rückgabemedium an die C Bridge, bzw. den SWF Interpreter (siehe Kapitel 5: Aufruf von COBOL).

Anmerkung zur Implementierung: Der COBOL Aufruf ist nur soweit unterstützt, dass die entsprechende C Bridge aufgerufen wird und dort die entsprechenden Änderungen im Java COBOL Objekt direkt vorgenommen werden, ohne vorher eine reale COBOL Funktion aufzurufen.

4 Ausführung eines Subworkflows

4.1 Starten der Subworkflowsteuerung

Der Carrier startet die Subworkflowsteuerung (SWFC), die anschließend die Finite State Machine (FSM) (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM)) startet und (sofern vorhanden) eine Konfigurationsdatei einliest und parst (siehe Kapitel 13.8: Beispiel-Konfigurationsdatei für die Subworkflowsteuerung). Anschließend startet der SWFC alle übrigen Module.

Um im Falle eines Systemcrash den Verlust von Informationen zu verhindern wird beim Start des SWFC geprüft, ob zum Zeitpunkt des Crashs ein Subworkflow (SWF) aktiv war. Um dies zu realisieren, gibt es zwei Ansätze:

- Beim optimistische Ansatz nimmt man an, dass Systemcrashs äußerst selten sind und versucht den „Normalfall“ für den Durchsatz zu optimieren. Hierbei wird zu Beginn eines jeden Subworkflows ein implizites beginTx ausgeführt, was sicherstellt, dass der Auftrag gesichert wird. Dies wird normalerweise über persistente Queues des Carriers realisiert. Somit ist sichergestellt, dass auch im Falle eines Systemcrashs der Auftrag (Subworkflow Controller Request Work Unit, SWFC RWU) abgesichert ist. Damit ist es ausreichend, beim Start des SWFC einmalig zu prüfen, ob ein alter Savepoint existiert und gegebenenfalls die Ausführung des Subworkflows an dieser Stelle mit den entsprechenden globalen und lokalen Daten wieder aufzunehmen, was mit einem entsprechenden außerordentlichen Zustandswechsel gekoppelt ist, um den SWF im richtigen Zustand fortsetzen zu können.

  Sollte kein alter Savepoint existieren wird entweder direkt die nächste SWFC RWU von der In Queue gelesen und bearbeitet (wenn der Carrier mit persistenten Queues arbeitet) bzw. geprüft, ob noch eine nicht bearbeitete SWFC RWU existiert und gegebenenfalls bearbeitet (wenn der Carrier nicht mit persistenten Queues arbeitet).

- Beim pessimistischen Ansatz nimmt man an, dass Systemcrashs häufig sind und versucht den „Recovery-Fall“ für den Durchsatz zu optimieren. Im Unterschied zum optimistischen Ansatz wird zusätzlich nach der erfolgreichen Bearbeitung jedes Subworkflows automatisch ein Savepoint geschrieben wird, damit bei einem Systemabsturz nach dem Ende des Subworkflows aber bevor die Ergebnis SWFC RWU in die Out Queue gelegt werden konnte, beim Neustart des SWFC die Ergebnis SWFC RWU direkt in die Out Queue gelegt werden kann und der SWF nicht erneut ausgeführt werden muss.

(In den nachfolgenden Use Cases wird der optimistische Ansatz mit dem Einsatz eines Carriers, der ohne persistente Queues arbeitet, beschrieben)

Anschließend startet der SWFC die FSM und die Endlos-Schleife, in der er über den I / O Controller auf ankommende Aufträge zur Verarbeitung wartet.

Siehe Use Case: 01, 02 und 03 in Kapitel 13.6: Use Cases.

Anmerkung zur Implementierung: Die Persistent Reusable Java Virtual Machine [JVM] ermöglicht es mehrere Java Virtual Machines (JVM) als ein JVM Set zu starten, so dass mehrere Virtual Machine Instanzen parallel ausgeführt werden. In diesem Umfeld ist es wichtig, dass eine Konfigurationsdatei pro JVM verfügbar ist.

4.2 Beenden der Subworkflowsteuerung

Der MFC signalisiert dem SWFC über eine spezielle SHUTDOWN SWFC RWU, dass er sich beenden soll. Daraufhin beendet der SWFC die FSM und informiert anschließend den MFC über das Notification API (siehe Kapitel 4.5: Senden von Nachrichten an den MFC (Notification API)) davon, dass er sich beendet. Anschließend informiert er alle Module davon, dass sie sich beenden sollen und beendet sich danach selbst.

Siehe Use Case: 04 in Kapitel 13.6: Use Cases.

4.3 Initialisierung eines Subworkflows

Der SWFC ist über den I / O Controller in der Lage, Aufträge (SWFC RWUs) vom MFC anzunehmen bzw. bearbeitete Ergebnisaufträge (Ergebnis SWFC RWUs) an den MFC zurückzugeben. Dies geschieht über zwei vom Carrier verwaltete Warteschlangen (Queues), eine Eingabe-Warteschlange (In Queue), in der vom MFC ankommende Aufträge als SWFC RWU auf Verarbeitung warten, bzw. eine Ausgabe-Warteschlange (Out Queue), über die verarbeitete Aufträge (Ergebnis SWFC RWUs) an den MFC zurückgegeben werden (Abbildung 9). Alternativ kann der SWFC auch über eine WSDL-Schnittstelle angesprochen werden (Abbildung 10).

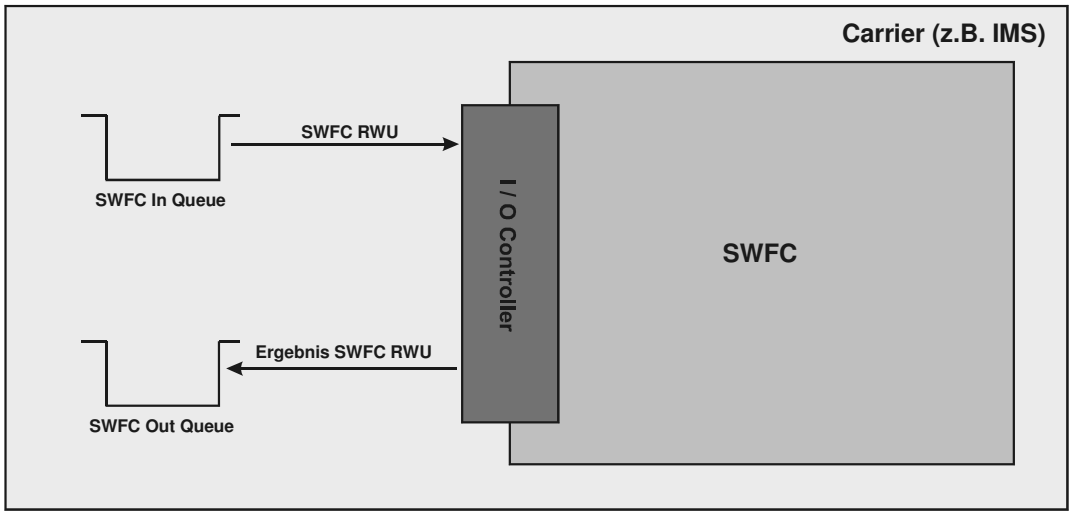

Abbildung 9: SWFC In und Out Queue

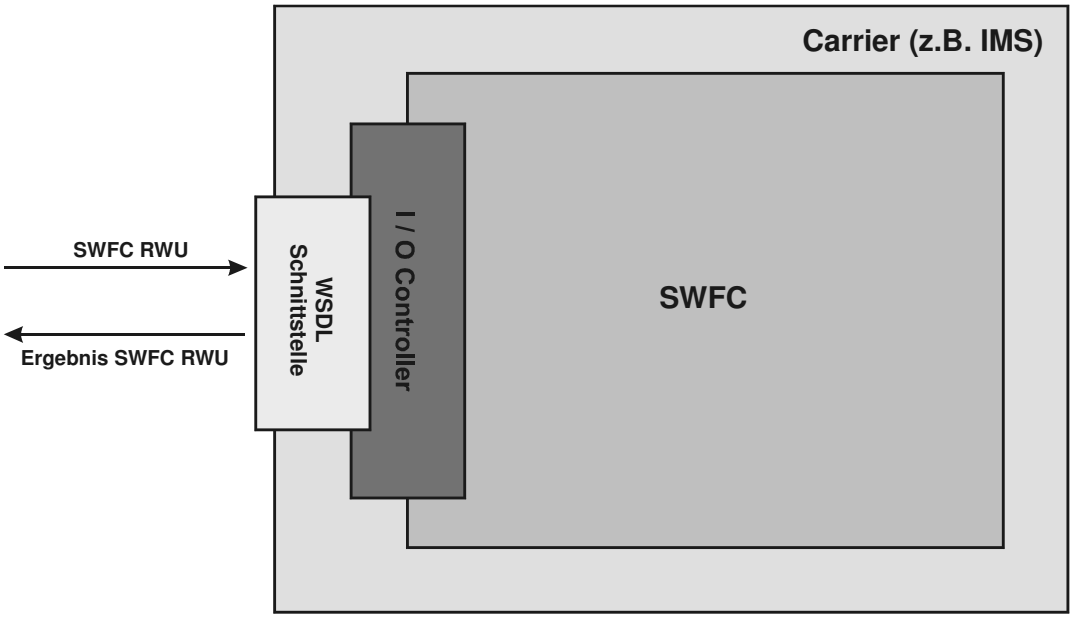

Abbildung 10: WSDL Schnittstelle

Ein Auftrag (SWFC RWU) besteht aus globalen Ein- und zusätzlichen globalen Rückgabefeldern, die jedoch inhaltlich undefiniert sind, da sie erst im SWF gefüllt werden. Somit kann die Eingabe und Ausgabe strukturell als gleichwertig betrachtet werden, wobei die ersten zwei Felder immer die SWF ID und die SWF Versionsnummer beinhalten (Abbildung 11).

Abbildung 11: Struktur der SWFC RWU

Der I / O Controller überführt die SWFC RWU in ein Java RWU Objekt und übergibt dieses an den SWFC. Dieser übergibt dann die SWF ID und SWF Versionsnummer an den SWFDL Parser, der dann anhand dieser Information mit Hilfe des Data Service die entsprechende SWFDL-Beschreibung einliest, parst und daraus eine SWF Repräsentation (z.B. ein Java Objektgraph) erzeugt. Diese SWF Repräsentation wird zusammen mit dem Message Container, der die benötigten Daten für diesen Subworkflow verwaltet (siehe Kapitel 4.4: Verwaltung der globalen und lokalen Daten), an den SWF Interpreter übergeben, der den zu diesem Auftrag gehörigen Subworkflow bearbeitet.

Um zu vermeiden, dass der SWFDL Parser bei jedem Start eines SWF die SWFDL-Beschreibung parsen muss, werden bereits generierte SWF Repräsentationen zwischengespeichert.

Siehe Use Case: 06, 07, 08 und 09 in Kapitel 13.6: Use Cases.

Anmerkung zur Implementierung: Der I / O Controller läuft im Augenblick nur unter einer angepassten Version des in [BEY] beschriebenen C Launchers direkt unter z/OS als Carrier, unterstützt also insbesondere nicht den Zugriff über eine Web Service Schnittstelle.

4.4 Verwaltung der globalen und lokalen Daten

4.4.1 Erzeugung des Message Containers

Der I / O Controller überführt die SWFC RWU in ein Java RWU Objekt welches an die Container Factory weitergegeben wird. Die Container Factory erzeugt dann aus diesem Java RWU Objekt mit Hilfe der oben erwähnten Data Section Copystruktur den Message Container, dessen fehlende globale und lokale Daten mit Hilfe der Data Description und dem Data Service aus 1..n Datenbanken gefüllt werden. Dieser Message Container verwaltet die globalen und lokalen Daten und stellt für die Aktivitäten ein mengenorientiertes Datenmanipulation API auf einer lokalen Sicht für die jeweilige Aktivität zur Verfügung.

4.4.2 Ausführung einer SWF Aktivität

Die SWFDL-Beschreibung definiert die Daten, die eine Aktivität liest bzw. schreibt über Ein- (Input Message) und Ausgabe-Signaturen (Output Message). Somit ist nach dem Parse-Vorgang für jede Aktivität bekannt, welche globalen bzw. lokalen Daten sie für die Ausführung benötigt bzw. verändert.

Die Ausführung einer SWF Aktivität erfolgt in drei Phasen: Initiate SWF Activity, Execute SWF Activity, Release SWF Activity:

- In der Initiate SWF Activity-Phase wird mit Hilfe der Message Signatur bei einer Java SWF Aktivität eine lokale Sicht des Message Containers erstellt und bei einer COBOL Aktivität ein Java COBOL Objekt, die an die Aktivität beim Aufruf als Parameter übergeben wird. Diese lokale Sicht ermöglicht nur den Zugriff auf die für die Aktivität benötigten Daten über ein mengenorientiertes Datenmanipulation API (Abbildung 12),
- In der Execute SWF Activity-Phase wird die Aktivität ausgeführt,
- In der Release SWF Activity-Phase werden bei Java SWF Aktivitäten die eventuell geänderten Daten in der lokalen Sicht und bei COBOL SWF Aktivitäten die eventuell geänderten Daten im Java COBOL Objekt wieder mit den Daten im Message Container synchronisiert. Des Weiteren werden in dieser Phase die für diese Aktivität registrierten Plug-Ins ausgeführt und anhand des Return Codes der Aktivität die weitere Steuerung des Subworkflows beeinflusst (so wird z.B. beim Return Code RC_RESTORE, der SWF auf den letzten Savepoint zurückgesetzt – siehe Kapitel 8: Transaktionale Steuerung des Subworkflows).

Abbildung 12: Beispiel-Anfrage an eine lokale Sicht des Message Containers

Siehe Use Case: 10 in Kapitel 13.6: Use Cases.

4.4.3 Rückgabe der Ergebnis SWFC RWU an den MFC

Der Message Container kann die globalen Daten als Java RWU Objekt ausgeben. So wird am Ende des Subworkflows aus dem Message Container ein Java RWU Objekt erzeugt, welches dann wieder an den SWFC als Resultat des fehlerfreien Subworkflows zurückgegeben wird. Dieser gibt das Java RWU Objekt an den I / O Controller weiter, der dieses wieder in das entsprechende Format für die Out Queue überführt und das Resultat (die Ergebnis SWFC RWU) in die Out Queue legt.

Siehe Use Case: 11 in Kapitel 13.6: Use Cases.

4.4.4 Persistierung der globalen und lokalen Daten

Der Message Container ist in der Lage, geänderte Daten wieder in die jeweilige Datenbank rückzuschreiben (z.B. bei commitTx). Hierfür bietet der Message Container ein

einfaches API an, wobei der eigentliche Datenbankzugriff über den Data Service erfolgt (Abbildung 13).

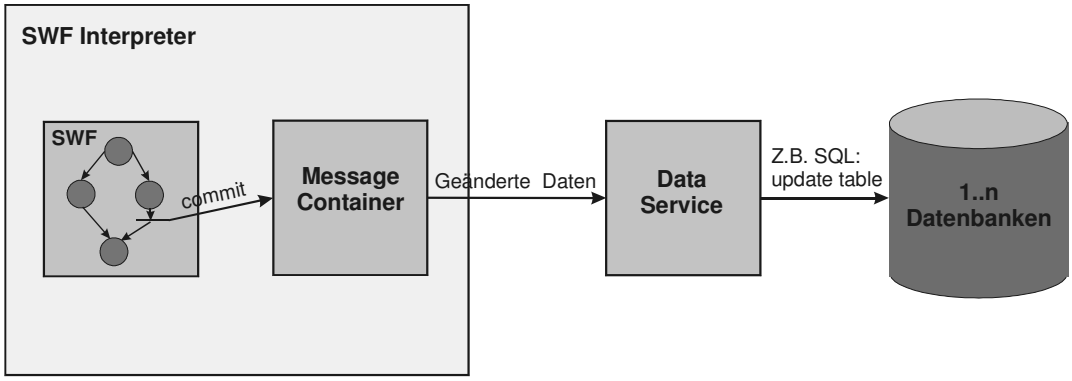

Abbildung 13: Beispiel für commitTx über Message Container

Davon abgesehen, bietet der Message Container die Möglichkeit die globalen und lokalen Daten (inklusive zusätzlicher Statusinformationen für den SWF Interpreter) an Savepoints zu persistieren bzw. auf einen beliebigen vorherigen Savepoint zurückzusetzen. Diese Savepoint Informationen können z.B. in eine eigene SWFC Datenbank geschrieben werden oder über eine Datei persistiert werden (Abbildung 14).

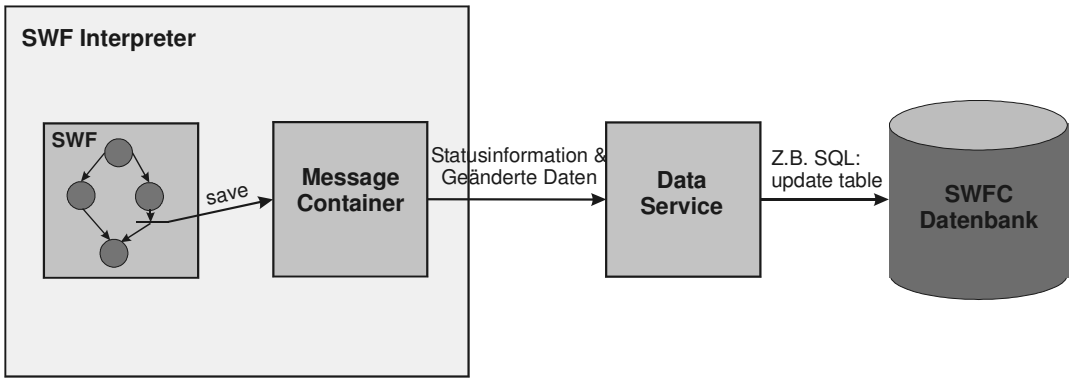

Abbildung 14: Beispiel für Savepoint über Message Container

Siehe Use Case: 12 und 13 in Kapitel 13.6: Use Cases.

4.4.5 COBOL API des Message Containers

Der Message Container ermöglicht es über das COBOL API die globalen und lokalen Daten, die eine COBOL Aktivität benötigt, als Java COBOL Objekt auszugeben. Dieses Java COBOL Objekt ist ein Datenobjekt, welches die Umsetzung des COBOL Aufrufes analog wie in [KOO] beschrieben ermöglicht. Außerdem ermöglicht das COBOL API das Synchronisieren von geänderten Daten aus diesem Java COBOL Objekt mit den

globalen und lokalen Daten im Message Container (siehe Kapitel 5: Aufruf von COBOL).

Siehe Use Case: 14 in Kapitel 13.6: Use Cases.

4.5 Senden von Nachrichten an den MFC (Notification API)

Das Notification API erfüllt zwei Funktionen:

- Es bietet dem SWF Interpreter die Möglichkeit den MFC von aufgetretenen Fehlern (siehe Kapitel 7: Fehlerbehandlung des Subworkflows und Abbildung 15) zu informieren,

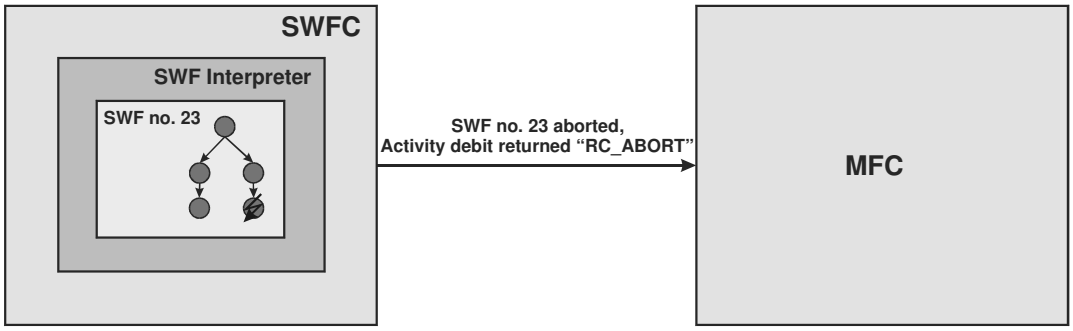

Abbildung 15: Beispiel für das Benachrichtigen des MFC im Fehlerfall (RC_ABORT)

- Und es bietet den Java Aktivitäten eine Möglichkeit zum Senden von Nachrichten an den MFC, so könnte z.B. bei einer credit-debit-Anwendung eine Nachricht nach der erfolgreichen Ausführung der credit-Operation aus der Aktivität und eine Nachricht nach der erfolgreichen Ausführung der debit-Operation aus der anschließenden Aktivität „gesendet“ werden (Abbildung 16).

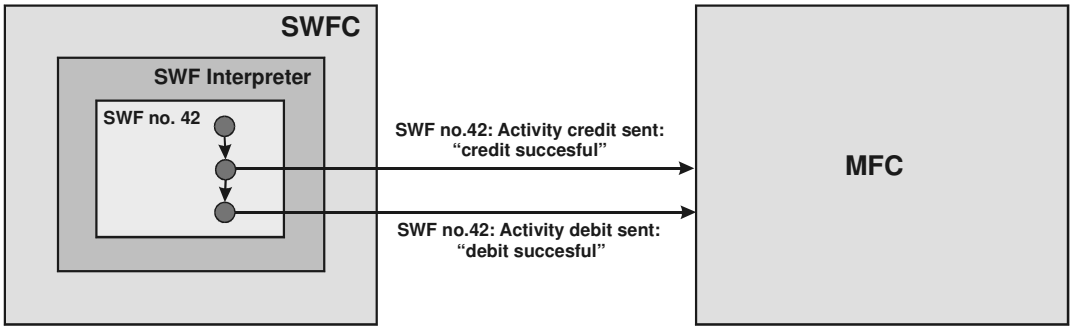

Abbildung 16: Beispiel für das Senden einer „beliebigen“ Nachricht an den MFC

Dieses Notification API wird vom I / O Controller zur Verfügung gestellt, wobei senden hier bedeutet, dass der I / O Controller die vom SWF Interpreter bzw. einer Aktivität erhaltene Nachricht in die Out Queue bzw. eine spezielle Notification Queue legt. Dabei gilt, dass es dem MFC überlassen ist, wie er auf die Fehlernachrichten reagiert bzw. ob er die von Aktivitäten initiierten Nachrichten (z.B. Meldung über erfolgreiche Ausführung einer Aktivität) an den eigentlichen Auftraggeber weiterleitet oder verwirft.

Falls der Auftrag über die Web Service Schnittstelle angenommen wurde, leitet der I / O Controller die Fehler- bzw. Aktivitäten-Nachrichten direkt an den Auftraggeber weiter.

5 Aufruf von COBOL Aktivitäten

5.1 Überblick

Die Subworkflowsteuerung (SWFC) unterstützt Java und COBOL Aktivitäten. Hierbei können Java Aktivitäten direkt ausgeführt werden, da die Subworkflowsteuerung in Java implementiert ist. Da Java aber bislang keine Mechanismen zum direkten Aufruf von COBOL Aktivitäten unterstützt, wird dieser Aufruf mit Hilfe einer im Nachfolgenden beschriebenen C Bridge realisiert.

5.2 Erforderliche Initialisierungen

Java ermöglicht es über einen Java Native Interface (JNI, siehe [LIA]) Call C Funktionen aufzurufen. Somit kann eine COBOL Aktivität ausgeführt werden, in dem eine C Funktion aufgerufen wird, die wiederum die COBOL Aktivität aufruft und entsprechende Konversionen von Java nach C nach COBOL bzw. vice versa ausführt.

Der Source Code dieser C Funktion wird offline in einem Parse-Step über die SWFDL-Definitionen erzeugt und anschließend im Compile-Step kompiliert, wobei alle C Funktionen, die für einen SWF relevant sind, in einer Bibliothek zusammengefasst werden. Diese C Bibliothek wird nachfolgend als C Bridge bezeichnet (Abbildung 17).

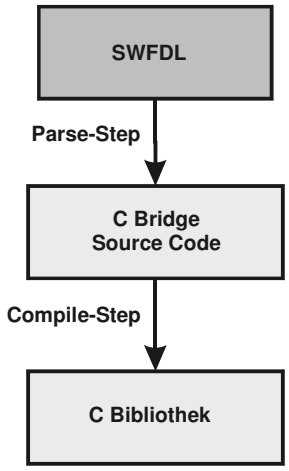

Abbildung 17: Parse-Step und Compile-Step für COBOL-Aufruf

Anmerkung zur Implementierung: Das Erstellen der C Bridge für den COBOL-Aufruf muss augenblicklich manuell erfolgen.

5.3 Aufruf der COBOL Aktivität

In der Initiate SWF Activity-Phase wird aus dem Message Container mit Hilfe der Message Signatur über das COBOL API ein Java COBOL Objekt erzeugt (Abbildung 18-1 und Abbildung 18-2). Anschließend wird in der Execute SWF Activity-Phase mit Hilfe der COBOL / JNI Facility die COBOL Aktivität aufgerufen (Abbildung 18-3). Diese benötigt den Namen der C Bridge, den Namen der COBOL Aktivität und das entsprechende Java COBOL Objekt als Parameter, um die C Bridge aufrufen zu können (Abbildung 18-4). In der C Bridge wird dann der Datenpuffer für die COBOL Aktivität über das Java COBOL Objekt initialisiert und die COBOL Aktivität aufgerufen, der als Parameter der gefüllte Datenpuffer übergeben wird (Abbildung 18-5). Nach der Ausführung der COBOL Aktivität in der C Bridge wird das Java COBOL Objekt wieder mit den geänderten Daten im Datenpuffer synchronisiert (Abbildung 18-6 und Abbildung 18-7) und an den SWF Interpreter zurückgegeben (Abbildung 18-8). Zum Schluss wird dann in der Release SWF Activity-Phase das u.U. geänderte Java COBOL Objekt mit den globalen und lokalen Daten im Message Container synchronisiert (Abbildung 18-9).

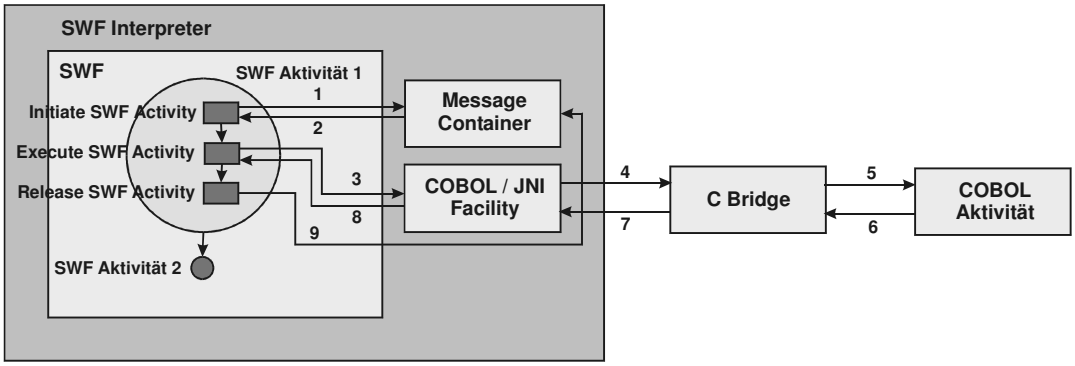

Abbildung 18: Aufruf einer COBOL Aktivität

Siehe Use Case: 14 in Kapitel 13.6: Use Cases.

Anmerkung zur Implementierung: Der COBOL Aufruf ist nur soweit unterstützt, dass die entsprechende C Bridge aufgerufen wird und dort die entsprechenden Änderungen im Java COBOL Objekt direkt vorgenommen werden, ohne vorher eine reale COBOL Funktion aufzurufen.

6 Plug-In Aktivitäten

6.1 Überblick

Um eine modulare und erweiterbare Konfiguration der Subworkflowsteuerung (SWFC) zu ermöglichen, kann nachträglich die Systemfunktionalität über sogenannte Plug-In Aktivitäten erweitert werden. So könnte z.B. eine Billing Plugin-Aktivität implementiert werden, die nach Ausführung jeder Aktivität eines Subworkflows den entsprechenden Preis berechnet und somit sicherstellt, dass nur die in Anspruch genommenen Leistungen bezahlt werden müssen.

Diese Plugin-Aktivitäten können sowohl in Java als auch in COBOL implementiert werden.

6.2 Einfügen von Plug-In Aktivitäten

Die Plug-In Aktivitäten können für eine SWF Aktivität „registriert“ werden, wobei die relevanten Daten (z.B. Trackinginformationen) vom Message Container verwaltet werden und somit nach Ausführung des Subworkflows dem MFC zur Verfügung stehen.

Plug-In Aktivitäten werden in SWFDL wie normale Aktivitäten definiert, mit dem Unterschied, dass sie nicht dem Interface (portType) des SWF angehören, sondern in einem eigenen Plug-In Interface (portType) definiert sind (Abbildung 19).

Anschließend können sie mit Hilfe des Verbs swfdl:plugin für eine SWF Aktivität registriert werden (siehe Kapitel 13.2: Subworkflow Definition Language (SWFDL) und 13.3.3: SWFDL-Beschreibung des Subworkflows), wobei hier das Namensattribut von swfdl:plugin mit dem Namensattribut der abstrakten Funktionsdeklaration (wsdl:operation) übereinstimmen muss.

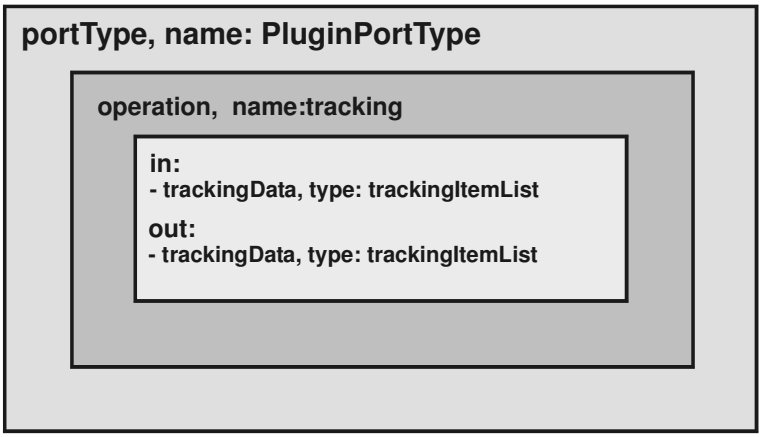

Abbildung 19: Beispiel Plug-In Interface für Tracking

6.3 Ausführung von Plug-In Aktivitäten

Der SWF Interpreter führt die Plug-In Aktivitäten nach erfolgreicher Beendigung einer SWF Aktivität (RC_OK) in der Reihenfolge, in der sie in der SWFDL-Beschreibung für die Aktivität registriert wurden, aus. Hierbei erfolgt die Ausführung der Plug-In Aktivitäten analog der Ausführung einer normalen Aktivität mit dem Unterschied, dass dieser zusätzlich der Name der gerade ausgeführten SWF Aktivität übergeben wird (Abbildung 20).

Dies führt aber im Gegensatz zu der Ausführung einer normalen SWF Aktivität nicht zu einem Zustandswechsel (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM)).

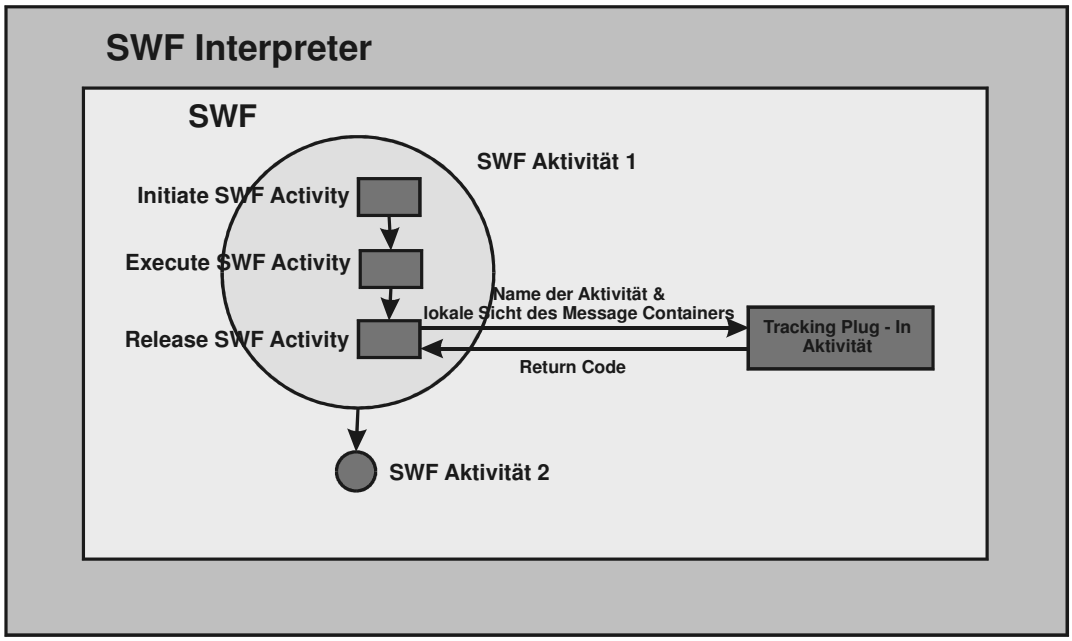

Abbildung 20: Beispiel für die Ausführung einer Java Plug-In Aktivität

Somit erfolgt die gesamte Ausführung der Plug-In Aktivität im Zustand Released SWF Activity des SWFC (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM)).

Über den Return Code als Ausgabeparameter kann eine Plug-In Aktivität auftretende Fehler bei der Ausführung an den SWF Interpreter signalisieren, wobei folgende Return Codes möglich sind:

- **RC_OK**: die Plug-In Aktivität wurde erfolgreich ausgeführt und der SWF Interpreter setzt die Ausführung des Subworkflows fort,
- **RC_RECOVER**: die Plug-In Aktivität wurde nicht erfolgreich ausgeführt und wird vom SWF Interpreter mit den gleichen Eingabewerten erneut gestartet,
- **RC_FAILED**: Die Plug-In Aktivität wurde nicht erfolgreich ausgeführt, der SWF Interpreter informiert den MFC über das Notification API (siehe Kapitel 4.5: Senden

von Nachrichten an den MFC (Notification API)) vom Scheitern des Aufrufes und setzt die Ausführung des Subworkflows fort.

Siehe Use Case: 15 in Kapitel 13.6: Use Cases.

7 Fehlerbehandlung des Subworkflows

7.1 Beschreibung

Jede Aktivität kann über ihren Rückgabewert signalisieren, ob und wenn welcher Fehler aufgetreten ist (siehe Kapitel 13.5: Liste der Return Codes). Der Vorteil dieser Strategie ist, dass die Aktivität aktiv das Verhalten des SWF Interpreter beeinflussen kann. Je nach Fehler ändert sich dann auch der Ausführungskontext (Execution Context) der entsprechenden ersten Aktivität, die nach dem Auftreten des Fehlers ausgeführt wird. Dies ermöglicht es, abhängig vom aktuellen Ausführungskontext der Aktivität, Entscheidungen zu treffen.

Generell können Fehler in zwei verschiedene Kategorien eingeteilt werden:

- Fehler, die von der Aktivität signalisiert werden:
    - Endet eine SWF Aktivität mit dem Return Code RC_RECOVER wird die Aktivität erneut mit denselben Eingabedaten und dem Ausführungskontext RECOVER gestartet (retry) und der MFC über das Notification API (siehe Kapitel 4.5: Senden von Nachrichten an den MFC (Notification API)) informiert, dass die Aktivität erneut gestartet wurde
    - Endet eine SWF Aktivität mit dem Return Code RC_ROLLBACKTX werden die globalen und lokalen Daten auf den Stand beim letzten commitTx bzw. beginTx zurückgesetzt, der SWF Interpreter setzt an dieser Stelle die Ausführung des SWF mit der ersten Aktivität nach dem letzten commitTx bzw. beginTx im Ausführungskontext ROLLBACKTX fort und der MFC wird über das Notification API informiert, dass ein rollbackTx ausgeführt wird. Sollte in der bisherigen Ausführung des Subworkflows bislang noch kein beginTx bzw. commitTx ausgeführt worden sein, dann wird der gesamte SWF mit denselben Eingabedaten neu gestartet,
    - Endet eine SWF Aktivität mit dem Return Code RC_RESTORE werden die globalen und lokalen Daten auf den Stand des letzten Savepoint zurückgesetzt, der SWF Interpreter setzt an dieser Stelle die Ausführung des SWF mit der ersten Aktivität nach dem Savepoint im Ausführungskontext RESTORETX fort und der MFC wird über das Notification API informiert, dass der SWF auf den letzten Savepoint zurückgesetzt wird. Sollte in der bisherigen Ausführung des Subworkflows bislang noch kein Savepoint geschrieben worden sein, dann wird der gesamte SWF mit denselben Eingabedaten neu gestartet,
    - Endet eine SWF Aktivität mit dem Return Code RC_ABORT wird der gesamte SWF abgebrochen und der MFC über das Notification API vom Abbruch des SWF informiert,

  - Endet eine SWF Aktivität mit dem Return Code RC_ERRORxxx wird die normale Ausführung des SWF abgebrochen, der MFC über das Notification API informiert, dass der Fehler RC_ERRORxxx aufgetreten ist und der entsprechende Error Workflow ausgeführt (Abbildung 6-Pattern 2). Generell gilt das mit der erfolgreichen Ausführung der letzten Aktivität des Error Workflows der gesamte SWF endet, es sei denn dies ist eine transaktionale restore- oder rollbackTx-Aktivität. In diesem Fall wird die Ausführung des Subworkflows wieder an der entsprechenden Stelle aufgenommen.

- Und Fehler, die vom System signalisiert werden:
  - Bei Abbruch des gesamten SWF durch das System, z.B. im Falle eines Systemabsturzes wird die Ausführung des SWF an der Stelle des letzten Savepoints mit den entsprechenden globalen und lokalen Daten an diesem Punkt wieder aufgenommen (sog. forward recovery) und der MFC wird über das Notification API informiert, dass forward recovery stattgefunden hat.

Schwere Fehler führen zum Abbruch des gesamten SWF ohne erneuten Neustart, der MFC wird über das Notification API informiert, welche Art von Fehler aufgetreten ist und die globalen und lokalen Daten in den 1..n Datenbanken werden auf die Werte vor Beginn der Ausführung des Subworkflows zurückgesetzt, um sicherzustellen, dass die Datenbanken konsistente Daten enthalten.

Beispiele:

- Korrupte SWFC RWU,
- Der Aufruf fehlerhafter Aktivitäten, also z.B. Aktivitäten, welche Divisionen durch Null ausführen, etc.

Siehe Use Case: 16 und 17 in Kapitel 13.6: Use Cases.

8 Transaktionale Steuerung des Subworkflows

8.1 Beschreibung

8.1.1 Einführung

> *Eine Transaktion ist eine atomare Operation: Die Transaktion wird entweder ganz oder gar nicht durchgeführt. Eine Transaktion ist die Zusammenfassung von mehreren Datei- oder Datenbankoperationen, die entweder:*
>
> - *erfolgreich abgeschlossen wird oder*
> - *die Datenbank unverändert lässt.* [SP1]

Der SWFC erweitert das Konzept der klassischen Transaktion auf Workflow-Ebene und realisiert ACID-Eigenschaften für die Subworkflows. Diese transaktionalen Subworkflows sind somit atomare Workflows: die Subworkflows werden entweder ganz oder gar nicht durchgeführt. Da BPEL4WS nicht die benötigten Verben für die Definition von transaktionalen Subworkflows unterstützt, erweitert SWFDL BPEL4WS um Verben für die Beschreibung des transaktionalen Kontextes. So wird der Beginn einer Transaktion mit dem Verb beginTx und das Ende einer Transaktion mit dem Verb commitTx markiert. Zwischen beginTx und commitTx können 1..n Aktivitäten ausgeführt werden, deren Änderungen erst beim commitTx nach außen hin sichtbar werden. Ein commitTx ist allerdings sehr zeitintensiv, da u.U. Änderungen in verschiedenen Datenbanken durchgeführt werden müssen. Die Anzahl der commitTxs zu erniedrigen birgt allerdings das Risiko, dass die n. Aktivität einer größeren Transaktion abbricht, was die Wiederholung aller n Aktivitäten nach sich zieht.

Um dieses Dilemma zu vermeiden, unterstützt der SWFC ein dem SAGA-Konzept (siehe [GAR]) ähnliches Verfahren, welches es ermöglicht zwischen beginTx und commitTx 1..m transaktionslokale Rücksetzungspunkte zu definieren, sogenannte Savepoints. Diese Savepoints führen nicht zum Rückschreiben der geänderten Daten in die entsprechenden Datenbanken, ermöglichen es aber trotzdem, im Fehlerfall auf den beliebig letzten Savepoint zurückzusetzen und mit der Ausführung des SWF an dieser Stelle fortzufahren. Dafür sind die Verben save (schreibe einen Rücksetzungspunkt) und restore mit dem Attribut whichSavepoint, welches es ermöglicht auf den n letzten Savepoint zurückzusetzen, vorgesehen (Abbildung 21).

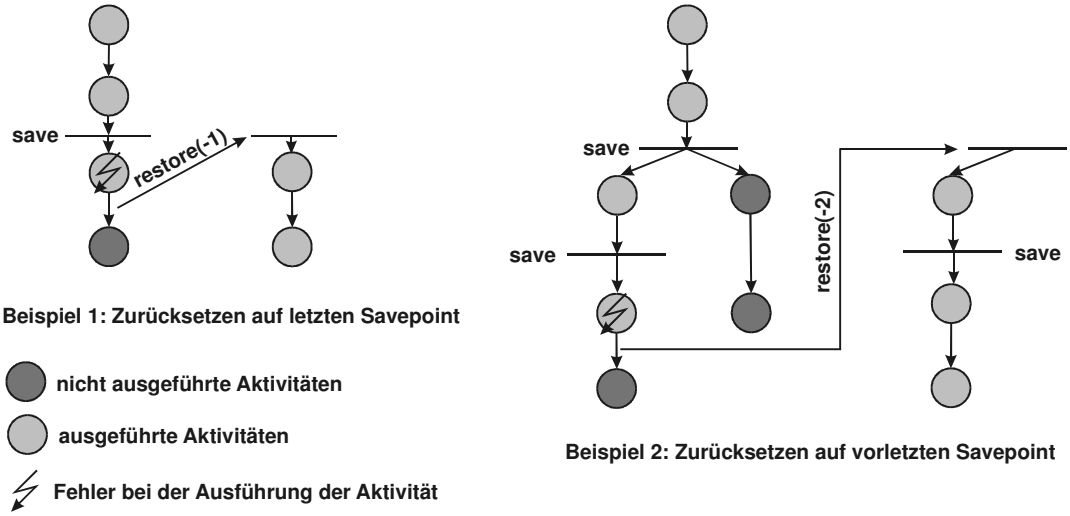

Abbildung 21: Savepoint-Beispiel

Im Falle, dass die gesamte Transaktion abgebrochen werden soll, ohne die Änderungen nach außen hin sichtbar zu machen, kann dies mit dem Verb rollbackTx signalisiert werden.

8.1.2 Umsetzung

Aus Gründen der Performance werden alle Änderungen der globalen und lokalen Daten nur lokal im Message Container durchgeführt und erst bei einem commitTx erfolgt ein Rückschreiben der geänderten Werte in die entsprechenden Datenbanken (siehe Abbildung 13). Dies führt dazu, dass der Message Container lokal Datenbankfunktionalität, wie Sicherung der Daten bei Savepoints bzw. das Rücksetzen der Daten bei Wiederherstellung von früheren Savepoints zur Verfügung stellt.

Bei der Sicherung von globalen und lokalen Daten an Savepoints übergibt der SWF Interpreter zusätzlich Statusinformationen, mit deren Hilfe die Ausführung des SWF im Wiederherstellungsfall an der entsprechenden Stelle wieder aufgenommen werden kann Abbildung 14).

Den Beginn einer Transaktion signalisiert der SWF Interpreter dem Message Container mit einem beginTx-Aufruf. Bei diesem Aufruf übergibt er dem Message Container die zusätzlichen Statusinformationen, womit dieser die globalen und lokalen Daten inklusive der Statusinformationen zu Beginn einer Transaktion persistieren kann, z.B. über einen Savepoint (Abbildung 22).

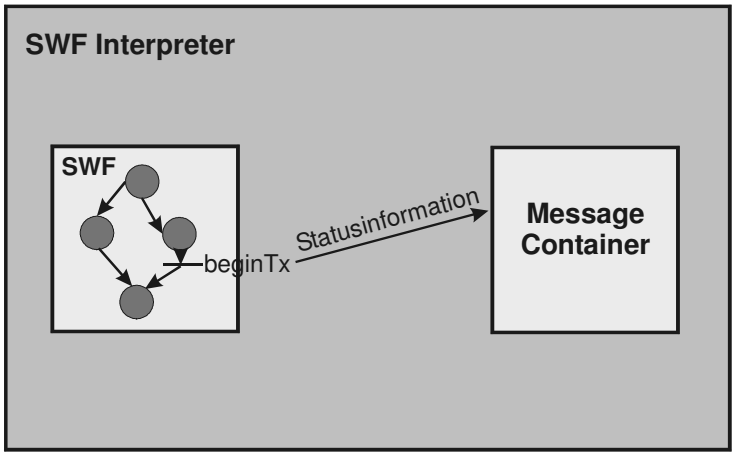

Abbildung 22: Beispiel für beginTx über Message Container

Genauso werden beim commitTx nicht nur die geänderten globalen und lokalen Daten in die Datenbanken rückgeschrieben, sondern auch Statusinformationen lokal persistiert, z.B. über einen Savepoint, wobei dies en block in eine eigene Datenbank erfolgt. Beim commitTx werden die Savepoint Daten ungültig, d.h. bei einem rollbackTx werden die Daten wie beim Start aus den Datenbanken initialisiert, da ansonsten die ACID Eigenschaften verletzt wären.

Falls bei der Ausführung des SWF ein Fehler auftritt, der die Rücksetzung der gesamten Transaktion erfordert, signalisiert der SWF Interpreter dies dem Message Container über einen rollbackTx-Aufruf und da die globalen und lokalen Daten zu Beginn und Ende jeder Transaktion persistiert wurden, kann der Message Controller diese auf den letzten gültigen Stand (entweder zum Zeitpunkt des letzten commitTx oder des letzten beginTx) zurücksetzen und liefert als Rückgabewert des Aufrufs die benötigten Statusinformationen, die der SWF Interpreter benötigt, um die Ausführung des SWF wieder an der richtigen Stelle aufnehmen zu können (Abbildung 23). Dieser rollbackTx-Aufruf kann entweder Teil eines Error Workflows sein oder eine Aktivität kann über ihren Return Code signalisieren, dass ein rollbackTx stattfinden soll (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM) und 7: Fehlerbehandlung des Subworkflows).

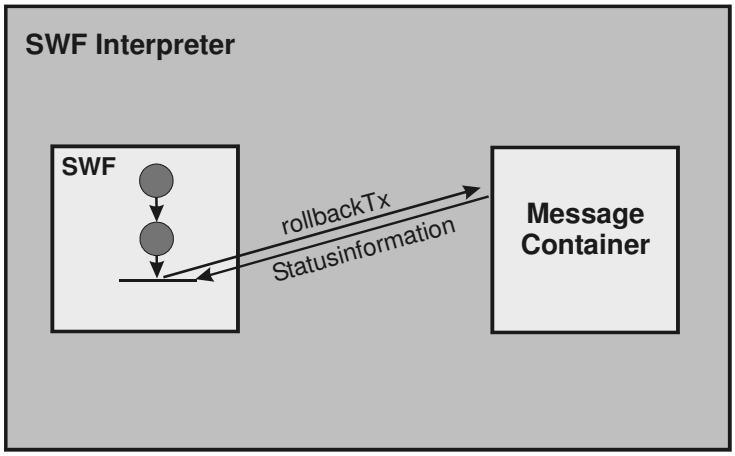

Abbildung 23: Beispiel für rollbackTx über Message Container

Alternativ kann an dieser Stelle, abhängig von der Fehlermeldung, sofern zwischendurch kein commitTx stattfand, auch nur auf den letzten Savepoint zurückgesetzt werden, wobei dieser restore-Aufruf ebenfalls entweder Teil eines Error Workflows sein kann, wo es dann auch möglich ist auf einen beliebigen der letzten vorhandenen Savepoints zurückzusetzen (Abbildung 21), oder von einer Aktivität über den Return Code signalisiert werden kann, wo dann automatisch auf den letzten vorhandenen Savepoint zurückgesetzt wird (siehe Kapitel 7: Fehlerbehandlung des Subworkflows).

8.1.3 Anmerkungen

- Der SWFC soll prinzipiell in der Lage sein, unter verschiedenen Carrier eingesetzt zu werden, wobei in dieser Diplomarbeit insbesondere drei verschiedene Fälle berücksichtigt werden: native, also direkt unter z/OS, unter IMS und unter CICS. Sofern der verwendete Carrier nicht mit persistenten Queues arbeitet (also nicht sichergestellt ist, dass Daten in Queue auch nach einem Systemabsturz wieder verfügbar sind), muss unmittelbar nach dem Zugriff des I / O Controller auf die In Queue ein implizites beginTx ausgeführt werden, welches sicherstellt, dass die SWFC RWU bei einem Systemcrash nicht verloren geht. Da IMS und CICS mit persistenten Queues arbeiten, d.h. ein implizites beginTx durchführen, ist dies nur im native Fall zu beachten.
- Der SWFC unterstützt keine expliziten Compensation Mechanismen, die z.B. für abgeschlossene Transaktionen (, die ein commitTx ausgeführt haben) Sinn machen würden, da die Ausführung des Subworkflows im Fehlerfall, wie in Kapitel 7 beschrieben beim letzten beginTx bzw. commitTx bzw. Savepoint wieder aufgenommen wird, und somit sichergestellt ist, dass der SWF auch im Fehlerfall vollständig ausgeführt wird. Wenn die Auswirkungen des Subworkflows (also die Änderungen der Daten in den 1..n Datenbanken) anschließend wieder rückgängig gemacht werden sollen, so muss dies über einen inversen separaten SWF erfolgen

(so könnte z.B. eine Überweisung über 100 € von A nach B mit einer Überweisung über 100 € von B nach A rückgängig gemacht werden).

- Außerdem wurde untersucht, in wie weit die von Butler et al. entwickelte Sprache StAC (siehe [BUT]) BPEL4WS (siehe [CUR]) als Basis für SWFDL ersetzen könnte. StAC ist ein Akronym für Structured Activity Compensation, eine Beschreibungssprache für verteilte, heterogene Umgebungen. Der Vorteil dieser Sprache gegenüber BPEL4WS ist, dass sie in Backus-Naur Form darstellbar und somit compilefähig ist und relativ einfach um transaktionale Verben, wie commitTx, rollbackTx, save und restore erweiterbar wäre. Allerdings erlaubt es StAC in der (zum Zeitpunkt der Erstellung dieser Diplomarbeit) aktuellen Version aus dem Jahr 2000 nicht, Aktivitäten bzw. den Datenfluss zu beschreiben, was eine essentielle Anforderung an die SWF Beschreibungssprache ist. Außerdem wird kein transaktionaler Kontext berücksichtigt, bzw. ist der Anwendungsbereich von StAC gedacht für verteilte Aktivitäten (über Carrier und Systemgrenzen), was beim SWF Pattern insbesondere im Hinblick auf Performance nicht gegeben ist.

Siehe Use Case: 18 und 19 in Kapitel 13.6: Use Cases.

9 Performance-Test und Bewertung

9.1 Überblick

Gemessen wurden drei verschiedene Testkonfigurationen:

- Workflow Patterns (siehe Kapitel 13.7.2.2): hier wurden fünf verschiedene Subworkflows, die jeweils eine Dummy-Aktivität ausführen, gemessen, um bestimmen zu können, wie viel Overhead durch die in Abbildung 6 dargestellten Workflow, bzw. den Aufruf einer COBOL-Aktivität entsteht,
- TPC™ A Benchmark (siehe Kapitel 13.7.2.1 und 13.7.2.3): hier wurde ein dem in [TPC] beschriebenen entsprechender Subworkflow mit verschiedenen Parametern gemessen, um bestimmen zu können, wie viel Overhead durch die transaktionale Subworkflowsteuerung gegenüber einer batch-orientierten Lösung entsteht. Als Vergleichsmessung diente eine angepasste Variante des in [BEY] beschriebenen Basic Online-Banking System (siehe Kapitel 13.7.2.1), welches eine entsprechende TPC A-Transaktion ausführte,
- Einkauf in einem Online Shop (siehe Kapitel 13.7.2.4): hier wurden sechs (Performance Messung Nr. 13 bis 15 entspricht einer Variante) verschiedene Varianten eines Subworkflows, der einen Einkauf in einem Online Shop modelliert, gemessen, um die Vorteile durch den Einsatz der transaktionalen Subworkflowsteuerung (z.B. Verkettung von mehreren Transaktionen zu einem Subworkflow) zu bestimmen.

Die Konfiguration des Testsystems, sowie die Beschreibung und Ergebnisse der einzelnen Messungen finden sich in Kapitel 13.7.

9.2 Bewertung

9.2.1 Workflow Patterns

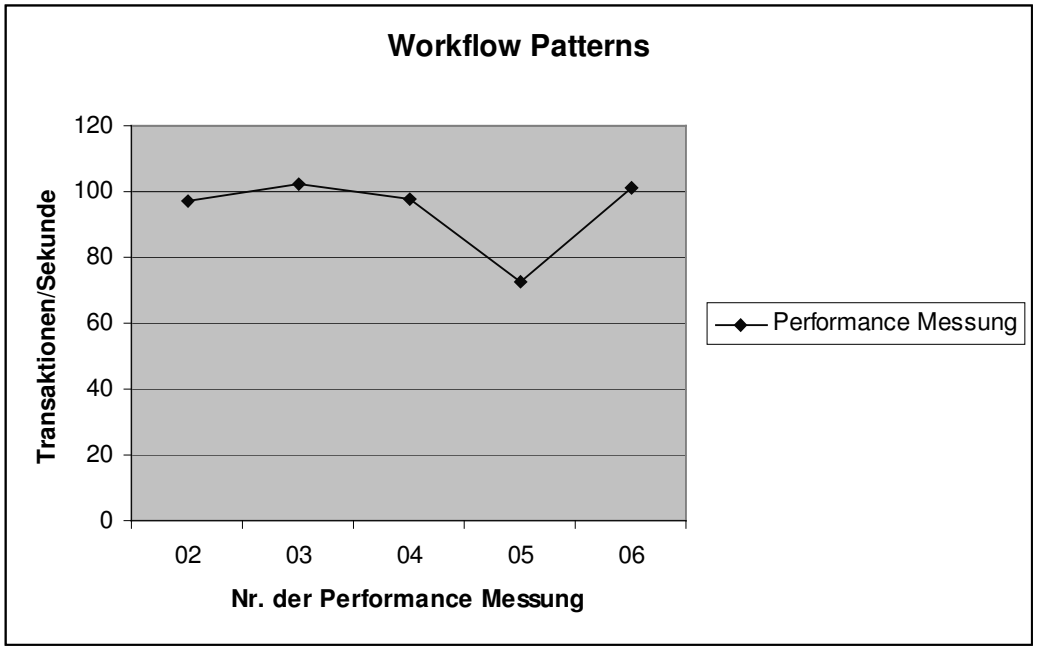

Tabelle 1: Vergleich der für die Workflow Patterns relevanten Performance Messungen

Die Auswertungen von Messung Nr. 02 bis 06 ergaben, dass die komplexeren Patterns, wie sequentiell, mit Fehler (Peformance Messung Nr. 03) oder sequentiell, mit Alternative (Performance Messung Nr. 04) keinen signifikanten Performance-Unterschied gegenüber dem Basis-Pattern sequentiell, ohne Fehler (Performance Messung Nr. 02) aufweisen. Insbesondere ergaben die Messungen, dass der Aufruf einer (simulierten) COBOL Funktion (Performance Messung Nr. 06) mit dem Aufruf einer Java Methode (Performance Messung Nr. 02) über Java Reflection zu vergleichen ist.

Das um ca. 25,1 % schlechtere Ergebnis des Patterns iterativ sequentiell (Performance Messung Nr. 05) zum Basis-Pattern (Performance Messung Nr. 02) erklärt sich damit, dass in dieser Zeit, die fünffache Anzahl der Ausgaben (bzw. Java Methoden Aufrufe über Reflection) geleistet werden.

9.2.2 TPC™ A Benchmark

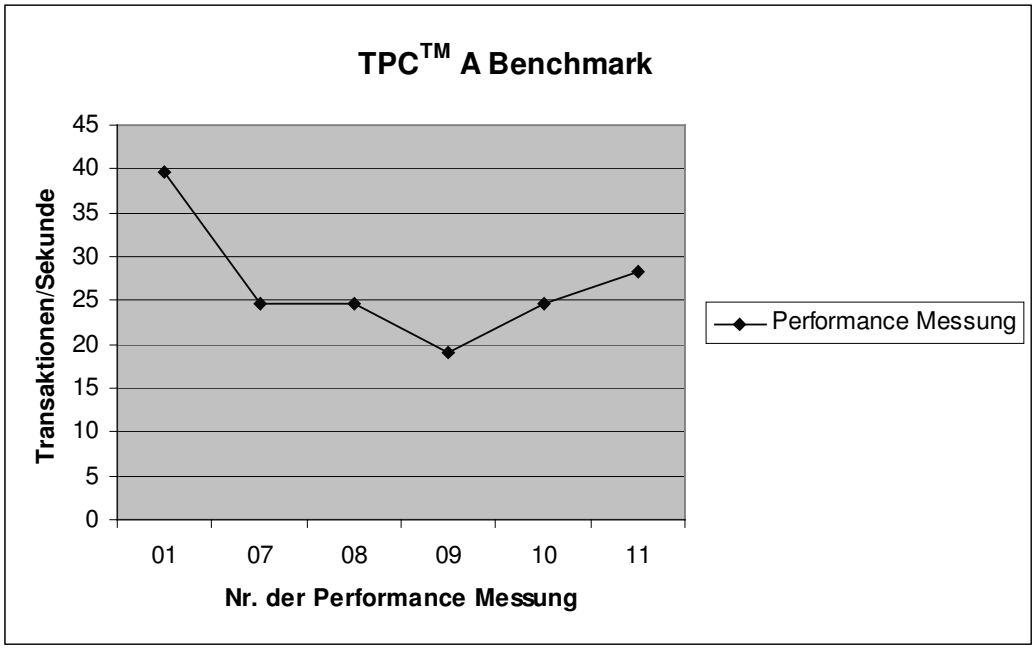

Tabelle 2: Vergleich der für den TPC™ A Benchmark relevanten Performance Messungen

Der Overhead durch den Einsatz der Subworkflowsteuerung (Performance Messung Nr. 07) zu der in [BEY] beschriebenen angepassten Version des Basic Online-Banking Systems (BOBS) (Performance Messung Nr. 01) beträgt ca. 38,0 %, wobei hier weder das Caching von Methoden (Performance Messung Nr. 08), noch das Ändern des Isolationslevels der Datenbank (Performance Messung Nr. 10) zu einer starken Verbesserung der Performance führten.

Der Versuch eine sinnvolle Messung ohne die Java Garbage Collection (Performance Messung Nr. 09, siehe [ARN]) durchzuführen, ließ sich mit den gewählten Einstellungen für die Persistent Reusable Java Virtual Machine (PRJVM) nicht realisieren, da der Hauptspeicher rapide bis an die (über die in Kapitel 13.9: Konfigurationsdatei der PRJVM, die bei den Performance-Messungen verwendet wurde dargestellte Konfigurationsdatei) maximal zugeteilte Menge von 256 Megabyte (MB) Hauptspeicher anstieg, wodurch die CPU Nutzung anstieg und die Performance stark absank.

Jedoch erbrachte der Einsatz einer alternativen Java Carrier Implementierung (Performance Messung Nr. 11) einen Performance-Gewinn von ca. 14,6 % gegenüber der in Kapitel 13.7.2.3.2 beschriebenen Basis Messung der Subworkflowsteuerung.

9.2.3 Einkauf in einem Online Shop

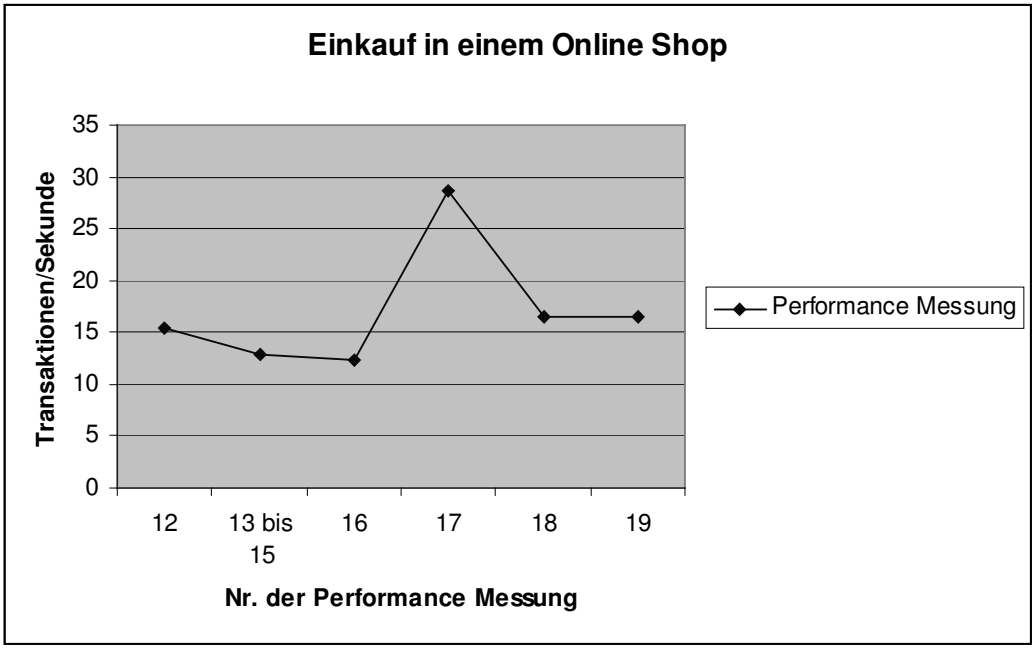

Tabelle 3: Vergleich der verschiedenen Performance Messungen für die Konfiguration Einkauf in einem Online Shop

Die Auswertungen von Messung Nr. 12 bis 19 zeigten, dass die Verkettung von drei Transaktionen in einem Subworkflow (Performance Messung Nr. 12) einen Performance-Gewinn von ca. 20,2 % gegenüber einer Stapelverarbeitungslösung (Performance Messung Nr. 13 bis 15) bzw. ca. 25,0 % gegenüber einer Lösung mit drei Subworkflows, die je eine der Transaktionen realisiert (Performance Messung Nr. 16), bedeutet.

Dazu hin zeigte sich, dass eine performante Realisierung des Data Services theoretisch einen Performance-Gewinn von ca. 85,2 % ermöglichen würde (Performance Messung Nr. 17), was darauf zurückzuführen ist, dass der Datenbankzugriff einen wesentlichen Bestandteil der Verarbeitungszeit eines Subworkflows darstellt.

Interessant anzumerken ist, dass die Abbruchszenarien (Performance Messung Nr. 18 und 19) um ca. 6,5 % schneller sind als die entsprechende Basismessung (Performance Messung Nr. 12).

9.2.4 Transaktionaler Subworkflow

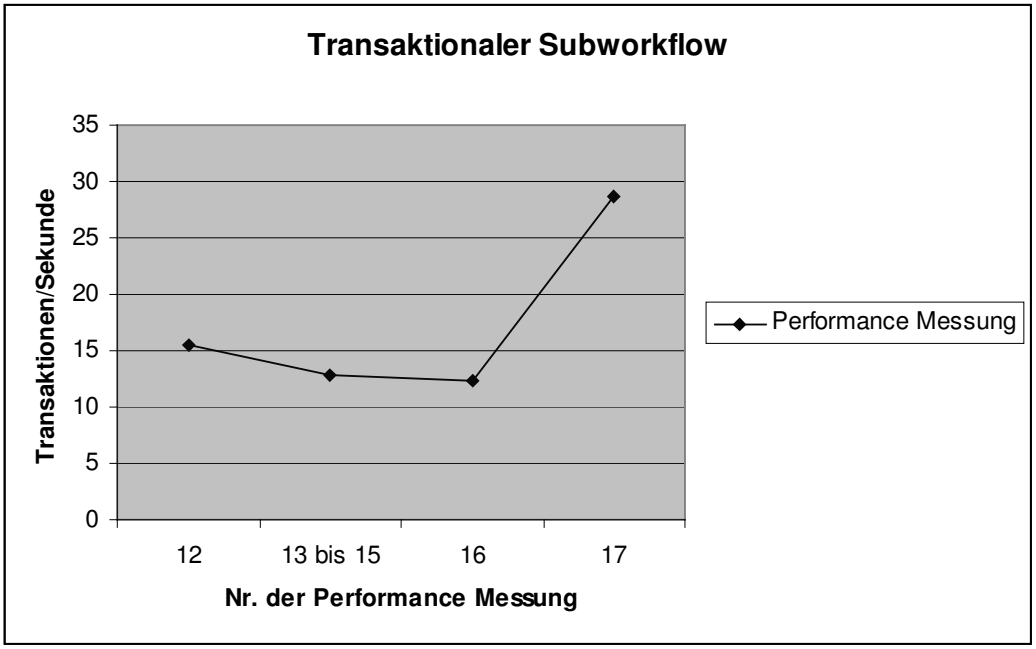

Tabelle 4: Vergleich der für den transaktionalen Subworkflow relevanten Performance Messungen

Die Auswertungen von Messungen Nr. 12 und Messungen Nr. 13 bis 15 zeigten den Vorteil des transaktionalen Subworkflow (Performance Messung Nr. 12) gegenüber Einzeltransaktionen (Performance Messung Nr. 13 bis 15), bzw. den Vorteil durch die Erweiterung von SWFDL um transaktionale Elemente (ca. 20,2 % Performance-Vorteil).

Die Auswertung von Performance Messung Nr. 17 zeigte das Potential für das Savepointing gegenüber commitTxs auf (ca. 122,5 % Performance-Vorteil gegenüber Einzeltransaktionen – Performance Messung Nr. 13 bis 15 – bzw. 85,2 % gegenüber der entsprechenden transaktionalen Variante – Performance Messung Nr. 16).

9.2.5 Zusammenfassung

In der Summe lassen sich folgende Schlussfolgerungen aufgrund der durchgeführten Performance-Tests treffen:

- Der Aufwand für die Ausführung der verschiedenen Workflow Patterns ist ungefähr gleich,
- Das Caching von Methoden führt nicht zu einem merklichen Performance-Gewinn,
- Der hauptsächliche Flaschenhals (Bottleneck) bei der Ausführung eines Subworkflows ist die Anbindung an die Datenbank,

- Der Einsatz eines hoch-performanten Transaktionsmonitors wie CICS oder IMS als Carrier dürfte zu deutlichen Performance-Steigerungen führen,
- Die Eigenschaft der Subworkflowsteuerung in einem Subworkflow mehrere Transaktionen verketten zu können, hat sich als klare Performance-Steigerung gegenüber der eine-Transaktion-pro-Subworkflow-Alternative erwiesen,
- Der Einsatz von transaktionslokalen Savepoints gegenüber commitTxs hat sich als klare Performance-Verbesserung gezeigt.

9.2.6 Offene Punkte

Die Performance-Messungen „offenbarten“ folgende offene Punkte der vorliegenden Implementierung der Subworkflowsteuerung, die aus Zeitgründen nicht mehr bearbeitet werden konnten:

- Der Hauptspeicher wächst linear mit der Anzahl der ausgeführten Subworkflows (siehe z.B. Abbildung 72),
- Es ist nicht möglich, mehrere Instanzen der Subworkflowsteuerung parallel als Java Virtual Machine (JVM) Workers in einem JVM Set (siehe [JVM]) zu starten, da die Subworkflowsteuerung beim Start von mehr als einem Worker sowohl den Namen des SAX Parsers als auch den Namen des JDBC Treibers als illegales Argument werten (IllegalArgumentException).
- Sowohl die Ausführung des Patterns sequentiell, mit Fehler (siehe Kapitel 13.7.2.2.3), als auch die der in Kapitel 13.7.2.4.6 beschriebenen Abbruch Szenarien verlaufen schneller als die entsprechenden Pendants ohne Fehler (siehe Kapitel 13.7.2.2.2 und 13.7.2.4.2).

10 Zusammenfassung und Ausblick

10.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde die Subworkflowsteuerung-Komponente der in [SP2] beschriebenen Transaktionsmaschine vollständig konzeptuell beschrieben und in großen Teilen implementiert. Diese Subworkflowsteuerung ermöglicht das Ausführen von Workflows mit ACID-Eigenschaften (sogenannten transaktionalen Workflows), wobei die einzelnen Aktivitäten (siehe [LE1]) entweder in Java oder in COBOL implementiert werden können.

Hierbei wurde zuerst die Funktionalität der einzelnen Module der Subworkflowsteuerung und anschließend anhand von Use Cases [JAC] die Interaktion dieser Module bzw. die Ausführung eines Subworkflows spezifiziert und über eine Finite State Machine (siehe Kapitel 13.4: Beschreibung der Finite State Machine (FSM) formalisiert (Grob-Design Phase). Darauf basierend wurde in der Fein-Design Phase die Umsetzung dieser Spezifikation auf UML Diagramme (siehe [JAC]) vorgenommen, eine XML-basierte (siehe [RAY]) Sprache (Subworkflow Definition Language, kurz SWFDL – siehe Kapitel 13.2: Subworkflow Definition Language (SWFDL)) für die Beschreibung der transaktionalen Workflows (hier Subworkflows genannt) und ein Programmiermodell für die Ausführung von Java und COBOL Aktivitäten (siehe Kapitel 3.2: Programmiermodell für Java und COBOL Aktivitäten) entwickelt. Danach wurde die Subworkflowsteuerung und die zugrunde liegende FSM in Java implementiert und abschließend wurde anhand von Performance-Messungen eine Bewertung dieser Implementierung vorgenommen (siehe Kapitel 9: Performance-Test und Bewertung).

Die vorliegende Implementierung ermöglicht es, einen Subworkflow in SWFDL zu spezifizieren und anschließend auszuführen, wobei die benötigten Daten entweder anfänglich über den zu bearbeitenden Auftrag mitgegeben oder aus einer Datenbank initialisiert werden können, wobei jedoch nur Java Aktivitäten unterstützt werden. Am Ende der Ausführung bzw. an jedem commit-Punkt werden alle geänderten Daten wieder in die Datenbank rückgeschrieben, bzw. ist es möglich über den Return Code einer Aktivität der Subworkflowsteuerung zu signalisieren, dass z.B. ein Rollback durchgeführt werden soll (siehe Kapitel 7: Fehlerbehandlung des Subworkflows und 8: Transaktionale Steuerung des Subworkflows). Die Ergebnisse der Bearbeitung des Auftrages werden als Ergebnisauftrag zurückgegeben.

Darüber hinaus wird eine Ausführung eines fehlerfreien Auftrags durch die Subworkflowsteuerung garantiert, so dass selbst im Falle eines Systemabsturzes sichergestellt ist, dass kein Auftrag verloren geht, bzw. ein inkonsistenter Zustand in der Datenbank entsteht.

10.2 Ausblick

Aufgrund des großen Umfangs der Funktionalität der Subworkflowsteuerung hätte es den zeitlichen Rahmen dieser Diplomarbeit gesprengt, die gesamte Funktionalität zu implementieren. Deshalb wurde in dieser Diplomarbeit der gesamten Funktionsumfang spezifiziert, aber nur ausgewählte Teile vollständig oder teilweise implementiert.

Somit sind mehrere Weiterführungen dieser Diplomarbeit denkbar, insbesondere die Umsetzung eines realen Geschäftsprozesses auf die Subworkflowsteuerung, die Umsetzung des I / O Controllers auf einen „realen“ Carrier (siehe Kapitel 1.3: Komponenten der Transaktionsmaschine)*,* wie CICS oder IMS bzw. eine hochperformante Implementierung des Data Service (siehe Kapitel 2.8: Module der Subworkflowsteuerung).

11 Akronyme

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Program Interface
BOBS	Basic Online-Banking System
BPEL4WS	Business Process Execution Language for Web Services
CICS	Customer Information Control System
ESCON	Enterprise System Connectivity
ESS	Enterprise Storage System
FSM	Finite State Machine
GB	Gigabyte
IMS	Information Management System
JCL	Job Control Language
JDBC	Java Database Connectivity
JNI	Java Native Interface
JVM	Java Virtual Machine
LPAR	Logical Partition
MB	Megabyte
MFC	Masterflow Controller
PRJVM	Persistent Reusable Java Virtual Machine
QoS	Quality of Service
RAS	Reliability, Availability, Serviceability
RMF	Resource Measurement Facility
SLA	Service Level Agreement
SQLJ	Embedded SQL in Java
SRWU	Service Request Work Unit
SUT	System Under Test
SWF	Subworkflow
SWFC	Subworkflow Controller
SWFC RWU	Subworkflow Controller Request Work Unit
SWFDL	Subworkflow Definition Language

TPC	Transaction Performance Council
UML	Unified Modeling Language
UoW	Unit of Work
WSDL	Web Services Description Language
XML	Extensible Markup Language

12 Literaturverzeichnis

12.1 Referenzen aus Zeitschriften und Büchern

[ARN] K. Arnold, J. Gosling: *The Java™ Programming Language, Second Edition.* Addison Wesley, 1998

[BEN] B. Bennett, B. Hahm, A. Leff, T. Mikalsen, K. Rasmus, J. Rayfield, I. Rouvellou: *A Distributed Object Oriented Framework to Offer Transactional Support for Long Running Business Processes.* Lecture Notes in Computer Science 1795, 08/2001

[BER] P. A. Bernstein, E. Newcomer: *Principles of Transaction Processing.* Morgan Kaufmann, 1997

[BEY] M. Beyerle: *Architekturanalyse der Java Virtual Machine unter z/OS und Linux.* Diplomarbeit, Fakultät für Informations- und Kognitionswissenschaften, Universität Tübingen, 2003

[BUT] M. Butler, C. Ferreira: *A Process Compensation Language.* Proceedings of the Integrated Formal Methods 2000 Conference, Lecture Notes in Computer Science, Vol. 1945, Springer-Verlag, 2000

[CHE] M. Chessell, C. Griffin, D. Vines, M. Butler, C. Ferreira, and P. Henderson: *Extending the concept of transaction compensation.* IBM Systems Journal, Vol. 41, No. 4, 2002.

[DA1] Datamonitor: *European corporate banking technology strategy.* Ref.-Code: BFTC0717, 05/2002

[DA2] Datamonitor: *Financial markets technology update, Driving IT Efficiencies.* Ref. – Code: BFTC0688, 02/2002

[DA3] Datamonitor: *Core Systems in European Retail Banking 2001-2005.* Ref.-Code: DMTC0838, 07/2002

[DAV] T. H. Davenport: *Process Innovation.* Harvard Business School Press, 1993

[ECK] B. Eckel: *Thinking in Java.* Prentice Hall, 1998

[FIS] M. Fisher, J. Ellis, J. Bruce: *JDBC API Tutorial and Reference, Third Edition.* Addison Wesley, 06/2003

[GA1] Gartner Inc.: *Gartner´s Application Development & Maintenance.* Research Note TG-12-9813, 1 March 2001

[GA2] Gartner Inc.: *From the Dustbin, Cobol Rises.* eWeek, May 28, 2001

[GAR] H. Garcia-Molina, K. Salem: *Sagas.* Proc. ACM Sigmod, 1987

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

[GRO] V. Grover, W. J. Kettinger: *Business Process Change.* Idea Group Publishing, 1995

[HAM] M. Hammer, J. Champy: *Reengineering the Corporation*. HarperBusiness, 1993

[HAR] B. C. Hardgrave, E. Reed Doke: *COBOL in an Object-Oriented World.* IEEE Software, Vol. 17, No. 2, March / April 2000, p.28

[JAC] I. Jacobson, G. Booch, J. Rumbaugh: *The Unified Modeling Language User Guide*. Addison-Wesley, April 2000, Chapter 16

[JVM] *New IBM Technology featuring Persistent Reusable Java Virtual Machines.* IBM Redbook SC34-6034-01

[KOO] A. L. Kooijmans et. al: *Java Programming Guide for OS/390: Chapter 15: The Java Native Interface.* IBM Redbook SG24-5619-00

[LE1] F. Leymann, D. Roller: *Production Workflow.* Prentice Hall, 2000

[LIA] S. Liang: *The Java Native Interface, Programmer´s Guide and Specification.* Addison Wesley, 1999

[MAR] J. Martin: *The Great Transition.* Amacom, 1995

[RAY] E. T. Ray: *Learning XML.* O´Reilly, January 2001

[SP1] W. G. Spruth, P. Hermann, U. Kebschull: *Einführung in z/OS und OS/390.* Oldenbourg-Verlag, 2003

[SP2] W. G. Spruth, J. Franz: *Reengineering von Kernanwendungssystemen auf Großrechnern.* Informatik Spektrum, Vol. 2, Mai 2003

[SP3] W.G. Spruth, U. Kebschull: *Kommerzielle Großrechner als Ausbildungsaufgabe an Universitäten und Fachhochschulen.* Informatik-Spektrum, 24. Juni 2001

[SP4] W.G. Spruth, J. Franz: *The Transaction Engine: High transaction rate processing for Core Business Applications.* Paper submitted for IEEE International Conference on Cluster Computing “Cluster 2003”. Intended for the Applications Chapter, Section Innovative Cluster Applications

[TPC] Transaction Performance Council (TPC): *TPC Benchmark™ A.* Standard Specification, Revision 2.0, 07 June 1994

[TUC] A. Tucker, R. Noonan: *Programming Languages, Principles and Paradigms.* McGraw-Hill, 2002

[WHI] Bill White, Rama Ayyar, Velibor Uskokovic: *zSeries HiperSockets.* IBM Redbook SG24-6816-00, 2002

[WIR] N. Wirth: *Compilerbau.* B.G. Teubner Verlag, Stuttgart 1984

12.2 Internet Referenzen

[CUR] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana: *Business Process Execution Language for Web Services, Version 1.1.*

http://www.ibm.com/developerworks/library/ws-bpel

[LE2] F. Leymann: *Web Services Flow Language, Version 1.0*.

http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[SPE] A. Z. Spector: *The IBM WebSphere Platform.*

http://www-3.ibm.com/developer/solutionsevent/pdfs/spector_lunchtime_keynote.pdf

13 Anhang

13.1 Inhaltsverzeichnis der CD

Nachfolgend wird die Verzeichnisstruktur der dieser Diplomarbeit beigelegten CD dargestellt. Hierbei werden Verzeichnisnamen *kursiv* dargestellt und der Grad der Einrückung entspricht der Tiefe im Verzeichnisbaum der CD:

/ausarbeitung		
	subworkflowsteuerung.pdf	Diese Ausarbeitung im pdf-Format
/literatur		
	/internet	Enthält Spiegelungen der Internet Referenzen
	/publikationen	Enthält ausgewählte referenzierte Publikationen
/source_code		
	/bobs	Enthält den angepassten Java und C Source Code des in [BEY] beschriebenen Basic Online-Banking Systems, bzw. die Konfigurationsdatei für die Persistent Reusable Java Virtual Machine (PRJVM)
	/swfc	
	/c/launcher	Enthält den C Source Code für den PRJVM Launcher
	/c/swfc	Enthält den SWFC-relevanten C Source Code
	/java/application	Enthält den Java Source Code für die in Kapitel 9: Performance-Test und Bewertung verwendeten Subworkflowaktivitäten
	/java/middleware	Enthält den Java Source Code der Subworkflowsteuerung (SWFC)
	/java/test_driver	Enthält den Source Code des in Kapitel 9: Performance-Test und Bewertung verwendeten Testtreibers
	/performance	Enthält das für die

	RMF-Performance-Messungen benötigte Post Processor Auswertungs-JCL-Skript
/property_files	Enthält die Konfigurationsdateien für die PRJVM und die Subworkflowsteuerung
/swfdl	Enthält die SWFDL-Beschreibungen für die in Kapitel *9:* Performance-Test und Bewertung verwendeten Subworkflows

13.2 Subworkflow Definition Language (SWFDL)

13.2.1 BPEL4WS-Erweiterungen für SWFDL

13.2.1.1 Überblick

SWFDL definiert folgende zusätzliche Verben zu der BPEL4WS-Spezifikation:

- swfdl:beginTx (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows),
- swfdl:binding: Dieses Verb ermöglicht es zu einer Interface-Beschreibung (wsdl:portType) Java, bzw. COBOL-spezifische Sprachdetails zu spezifizieren und ist dem in [LE2] vorgestellten Ansatz nachempfunden. Dies wird dadurch realisiert, dass zu jeder abstrakten Funktionsbeschreibung (wsdl:operation) eine konkrete Java-Methode (swfdl:javaOperation) bzw. eine konkrete COBOL dll und Funktion (swfdl:cobolOperation) spezifiziert werden kann,
- swfdl:commitTx (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows),
- swfdl:plugin (siehe Kapitel 6: Plug-In Aktivitäten),
- swfdl:restoreTx (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows und 7: Fehlerbehandlung des Subworkflows),
- swfdl:rollbackTx (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows und 7: Fehlerbehandlung des Subworkflows),
- swfdl:saveTx (siehe Kapitel 8: Transaktionale Steuerung des Subworkflows).

Jene Verben und Attribute, die für die Beschreibung von Subworkflows nicht benötigt werden, aber für eine BPEL4WS-Beschreibung essentiell sind, werden aus Gründen der Konformität mit dem BPEL4WS-Standard mit Dummy-Werten initialisiert, um SWFDL als syntaktisches Subset von BPEL4WS (mit entsprechenden Spracherweiterungen, die es z.B. ermöglichen Transaktionen zu beschreiben), zu definieren.

13.2.1.2 XML Schemas für die BPEL4WS-Spracherweiterungen

```
<?xml version="1.0"?>
<schema   xmlns="http://www.w3.org/2001/XMLSchema"
          xmlns:swfdl="http://www.thornung.com/2003/SWFDL"
          targetNamespace="http://www.thornung.com/2003/SWFDL">
  <complexType name="beginTxType">
  </complexType>
  <complexType name="commitTxType">
  </complexType>
  <complexType name="rollbackTxType">
  </complexType>
  <complexType name="saveType">
  </complexType>
  <complexType name="restoreType">
    <attribute name="whichSavepoint" type="string" />
  </complexType>
  <complexType name="pluginType">
    <attribute name="pluginName" type="string" />
  </complexType>
  <complexType name="bindingType">
    <sequence>
      <element name="function" type="functionType"
               minOccurs="1" maxOccurs="unbounded" />
    </sequence>
    <attribute name="name" type="string" />
    <attribute name="type" type="string" />
  </complexType>
  <complexType name="functionType">
    <group ref="swfdl:functionImplementation" minOccurs="1"
           maxOccurs="1" />
    <attribute name="name" type="string" />
  </complexType>
  <group name="functionImplementation">
    <choice>
```

```
        <element name="javaOperation" type="javaOperationType" />
        <element name="cobolOperation" type="cobolOperationType" />
      </choice>
    </group>
    <complexType name="javaOperationType">
      <attribute name="javaMethod" type="string" />
    </complexType>
    <complexType name="cobolOperationType">
      <attribute name="cobolDll" type="string" />
      <attribute name="cobolMethod" type="string" />
    </complexType>

    <element name="beginTx" type="beginTxType" />
    <element name="binding" type="bindingType" />
    <element name="commitTx" type="commitTxType" />
    <element name="plugin" type="pluginType" />
    <element name="restore" type="restoreType" />
    <element name="rollbackTx" type="rollbackTxType" />
    <element name="save" type="saveType" />
</schema>
```

13.2.2SWFDL Keywords

Anm.: Erweiterung zu BPEL4WS sind **fett** gekennzeichnet.

- process
  - wsdl:message
    - wsdl:part
  - wsdl:operation
    - wsdl:input
    - wsdl:output
  - wsdl:portType
  - bpws:getVariableData(‘variableName’, ‘partName’)
  - empty
  - invoke

    - catch
- sequence
- switch
    - case
    - otherwise
- terminate
- while
- **swfdl:binding**
    - **swfdl:function**
        - **swfdl:javaOperation**
        - **swfdl:cobolOperation**
- **swfdl:beginTx**
- **swfdl:commitTx**
- **swfdl:rollbackTx**
- **swfdl:saveTx**
- **swfdl:restoreTx**
- **swfdl:plugin**

13.2.3 Abstrakte Syntax

13.2.3.1 Definition

```
SWF = String name; Declarations decpart; Sequence body
Declarations =   PortType functionPortType;
                 [ PortType pluginPortType ];
                 Binding functionBinding; [ Binding pluginBinding ]
PortType = String name; ( Operation operation)*
Operation = String name; Message inputMessage, Message outputMessage
Message = String name; ( Part part )+
Part = String name; String type
Binding = String name; ( Function function )*
Function = JavaOperation javaOperation | CobolOperation cobolOperation
JavaOperation = String javaMethodName; String javaClassName
CobolOperation = String cobolFunctionName; String cobolDllName
```

```
Sequence = ( Statement statement )*
Statement =   Empty empty | Invoke invoke | While while |
              Switch switch | Terminate terminate |
              Sequence sequence | SaveTx saveTx |
              RestoreTx restoreTx | BeginTx beginTx |
              RollbackTx rollbackTx | CommitTx commitTx
Invoke =  String portType; String operation; ( Plugin plugin )*
          [ Sequence errorWorkflow ];
Plugin = String activityName; String operation
While = Condition test; Sequence body
Switch = ( Case branch )+ [ Otherwise alternativeBranch ]
Case = Condition test; Sequence body;
Otherwise = Sequence body
RestoreTx = int whichSavepoint
Condition = RelationalOp op; VariableData term1; Value term2
RelationalOp =   &lt; {<} | &lt;= {<=} | = {=} | != {!=} |
                 &gt; {>} | &gt;= {>=}
VariableData =   String variableName; String partName
                 {bpws:getVariableData( 'variableName', 'partName' )}
Value =   BoolValue boolValue | DoubleValue doubleValue |
          LongValue longValue | StringValue stringValue
BoolValue = boolean value
DoubleValue = double value
LongValue = long value
StringValue = String value
```

13.2.3.2 Anmerkungen

- [] markiert Elemente, die einmal oder keinmal vorkommen können
- ()* markiert Elemente, die beliebig oft (auch keinmal!) vorkommen können
- ()+ markiert Elemente, die mindestens einmal vorkommen können
- {} markiert Kommentare
- | entweder das Element links oder das Element rechts von | kann vorkommen (| fungiert als syntaktisches ODER)
- ; dient als Trennzeichen von Elementen

13.2.4 Semantik

13.2.4.1 Definition

```
M: SWF × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( SWF swf, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = M( swf.body, mc, s, dec, ec, fsm )

M: Sequence × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Sequence seq, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc                              wenn seq = Ø
   = for i = 1..n do M( (Statement) seq.statement_i, mc, s, dec, ec,
     fsm ) )
                                     wenn seq = statement_1..statement_n

M: Statement × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Statement st, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = M( (Empty) st, mc, s, dec, ec, fsm )        wenn s = Empty
   = M( (Invoke) st, mc, s, dec, ec, fsm )       wenn s = Invoke
   = M( (While) st, mc, s, dec, ec, fsm )        wenn s = While
   = M( (Switch) st, mc, s, dec, ec, fsm )       wenn s = Switch
   = M( (Terminate) st, mc, s, dec, ec, fsm )    wenn s = Terminate
   = M( (SaveTx) st, mc, s, dec, ec, fsm )       wenn s = SaveTx
   = M( (RestoreTx) st, mc, s, dec, ec, fsm )    wenn s = RestoreTx
   = M( (BeginTx) st, mc, s, dec, ec, fsm )      wenn s = BeginTx
   = M( (RollbackTx) st, mc, s, dec, ec, fsm )   wenn s = RollbackTx
   = M( (CommitTx) st, mc, s, dec, ec, fsm )     wenn s = CommitTx
```

```
   = M( (Flow) st, mc, s, dec, ec, fsm )        wenn s = Flow
   = M( (Sequence) st, mc, s, dec, ec, fsm )    wenn s = Sequence

M: Empty × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Empty e, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc

M: Invoke × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Invoke i, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = InvokeJavaOperation(  (JavaOperation)
                           dec.functionBinding.function.get(
                           i.operation ), i.operation, i.plugin,
                           i.errorWorkflow )
   wenn dec.functionBinding.function.get( i.operation ) = JavaOperation
   = InvokeCobolOperation( (CobolOperation)
                           dec.functionBinding.function.get(
                           i.operation ), i.operation, i.plugin,
                           i.errorWorkflow )
   wenn dec.functionBinding.function.get( i.operation ) =
   CobolOperation

M: While × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( While loop, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = while ( M( loop.test, mc, s, dec, ec, fsm ) ) do
     M( loop.body, mc, s, dec, ec, fsm )
```

M: Switch × MessageContainer × StatusInformation × Declarations ×
ExecutionContext × FSMState → MessageContainer × StatusInformation ×
ExecutionContext × FSMState

M(Switch sw, MessageContainer mc, StatusInformation s,
Declarations dec, ExecutionContext ec, FSMState fsm)

= M(sw.branch, mc, s, dec, ec, fsm) wenn M(sw.branch.test, mc, s, dec, ec, fsm) = true und dies der einzige Case-Zweig ist

= M($sw.branch_i$, mc, s, dec, ec, fsm) wenn M($sw.branch_i$.test, mc, s, dec, ec, fsm) = true und i = 1..n und es n Case-Zweige gibt

= mc wenn M($sw.branch_i$.test, mc, s, dec, ec, fsm) = false und i = 1..n und es n Case-Zweige gibt und kein alternativer Zweig (Otherwise) existiert

= M(sw.alternativeBranch, mc, s, dec, ec, fsm) wenn M($sw.branch_i$.test, mc, s, dec, ec, fsm) = false und i = 1..n und es n Case-Zweige gibt und ein alternativer Zweig (Otherwise) existiert

M: Terminate × MessageContainer × StatusInformation × Declarations ×
ExecutionContext × FSMState → MessageContainer × StatusInformation ×
ExecutionContext × FSMState

M(Terminate t, MessageContainer mc, StatusInformation s,
Declarations dec, ExecutionContext ec, FSMState fsm)

= beendet diesen SWF

```
M: SaveTx × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( SaveTx save, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc.saveTx( s, ec, fsm )

M: RestoreTx × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( RestoreTx restore, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc.restoreTx( restore.whichSavepoint )

M: BeginTx × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( BeginTx begin, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc.beginTx( s, ec, fsm )

M: CommitTx × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( CommitTx commit, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc.commitTx( s, ec, fsm )

M: RollbackTx × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( RollbackTx rollback, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
```

```
   = mc.rollbackTx()

M: Case × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Case c, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = M( c.body, mc, s, dec, ec, fsm )

M: Otherwise × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Otherwise o, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = M( o.body, mc, s, dec, ec, fsm )

M: Condition × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → MessageContainer × StatusInformation ×
   ExecutionContext × FSMState
M( Condition c, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = evaluateCondition( c.op, M( c.term1, mc, s, dec, ec, fsm ),
                        c.term2 )

M: VariableData × MessageContainer × StatusInformation × Declarations ×
   ExecutionContext × FSMState → Value
M( VariableData vd, MessageContainer mc, StatusInformation s,
   Declarations dec, ExecutionContext ec, FSMState fsm )
   = mc.getVariableData( vd.partName )

evaluateCondition: RelationalOp × Value × Value → boolean
evaluateCondition( RelationalOp op, Value v1, Value v2 )
   = v1 < v2             wenn op = &lt; und
                         v1, v2 != BoolValue | StringValue, andernfalls
```

```
                        wenn v1 oder v2 = BoolValue | StringValue, dann
                        false
= v1 <= v2              wenn op = &lt;= und
                        v1, v2 != BoolValue | StringValue, andernfalls
                        wenn v1 oder v2 = BoolValue | StringValue, dann
                        false
= v1 == v2              wenn op = =
= v1 != v2              wenn op = !=
= v1 > v2               wenn op = &gt; und
                        v1, v2 != BoolValue | StringValue, andernfalls
                        wenn v1 oder v2 = BoolValue | StringValue, dann
                        false
= v1 >= v2              wenn op = &gt;= und
                        v1, v2 != BoolValue | StringValue, andernfalls
                        wenn v1 oder v2 = BoolValue | StringValue, dann
                        false
```

13.2.4.2 Anmerkungen

- bpws:getVariableData(‘variableName’, ‘partName’) liefert den globalen oder lokalen Wert aus dem MessageContainer, der unter dem Schlüssel partName gespeichert ist. Dieses Konstrukt wurde aus Konformität mit dem ursprünglichen BPEL4WS in dieser Form belassen und wird benötigt, um Schleifen- bzw. Alternativ (switch)-Bedingungen in einem SWF auswerten zu können (Beispiel: <case condition=“bpws:getVariableData(‘creditCardInput’, ‘creditCardStatus’ != ‘STATUS_OK’”>),
- MessageContainer, StatusInformation, Declarations, ExecutionContext, FSMState werden nicht explizit bei jeder Interpretation eines Sprachelements mitübergeben, sondern stehen als Umgebungsvariablen zur Verfügung,
- StatusInformation speichert die Reihenfolge der Ausführung von Aktivitäten, um somit im Fall eines restoreTx bzw. rollbackTx die Ausführung des SWF wieder an der entsprechenden Stelle aufnehmen zu können,
- Declarations enthält die für die Ausführung von Plugins und Aktivitäten relevanten Informationen,
- ExecutionContext bezeichnet den aktuellen Ausführungskontext (siehe Kapitel 7: Fehlerbehandlung des Subworkflows),
- FSMState bezeichnet den aktuellen Zustand der Finite State Machine (Kapitel 13.4: Beschreibung der Finite State Machine (FSM)),

- InvokeJavaOperation(...) führt mit den entsprechenden zusätzlichen Informationen aus Declarations eine Java Aktivität wie in Kapitel 4.4.2: Ausführung einer SWF Aktivität beschrieben aus,
- InvokeCobolOperation(...) führt mit den entsprechenden zusätzlichen Informationen aus Declarations eine COBOL Aktivität wie in Kapitel 5: Aufruf von COBOL beschrieben aus.

Anmerkung zur Implementierung: Die aktuelle Implementierung beschreibt die Ausführung eines Subworkflows (siehe StatusInformation) über eine Folge von Indizes. Da allerdings bei einer Schleife (z.B. while flag != true) nicht sichergestellt werden kann, dass sie an der richtigen Stelle wieder aufgenommen wird, ist es nicht legal innerhalb einer Schleife einen Savepoint zu schreiben, bzw. eine Transaktion zu beenden oder zu starten (beginTx bzw. commitTx). Aus diesem Grund liefert der Parser einen Fehler beim parsen von SWFDL-Beschreibungen, bei denen dies versucht wird. RollbackTx- und restoreTx-Aufrufe innerhalb einer Schleife sind jedoch legal.

13.2.5 WSDL-SWFDL Vergleichsmatrix

WSDL	Beschreibung	SWFC
Binding	Protokollspezifische Details der Implementierung	X
Message	Potentielle Input, Output oder Fault Signatur einer Operation	X
Operation	Funktion	X
Port	Assoziiert ein Binding mit einem spezifischen Netzwerkendpunkt	X
PortType	Interface	X
Service	Sammlung von Ports	X
ServiceType	Sammlung von PortTypes	X
Type	Beschreibung von komplexen Datentypen	X

Tabelle 5: WSDL-SWFDL Vergleichsmatrix

13.2.6 BPEL4WS-SWFDL-Vergleichsmatrix

BPEL4WS	Beschreibung	SWFC
Assign	Ermöglicht Zuweisungen von Messages in verschiedenen Con-	-

	tainern	
bpws:getContainerData(‘containerName’, ‘partName’)	Liefert den aktuellen Wert von containerName.partName als Ergebnis (es gibt immer nur einen Message-Typ pro Container)	X
Catch	Fault Handler, der einen spezifischen Fehler behandelt	X
CatchAll	Fault Handler, der alle Fehler behandelt	-
Compensate	Ermöglicht Kompensation einer Aktivität	-
CompensationHandler	Wrapper für Compensation Aktivitäten	-
Container	Speicher für Nachrichten	X
Containers	Sammlung von Container(n). Hier werden Container definiert	X
CorrelationSet	Ermöglicht es, Nachrichten miteinander in Verbindung zu bringen. Interessant für länger dauernde asynchrone Kommunikation	-
CorrelationSets	Sammlung von CorrelationSet	-
Empty	Noop	X
FaultHandlers	Ermöglicht es eine Sammlung von catch-Statements zu definieren, die jeweils für eine bestimmte Art von Fehler zuständig sind	-
Flow	Parallelverarbeitung	K
Invoke	Aufruf einer Aktivität	X
Partner	Die Teilnehmer an einem Geschäftsprozess werden über Rollen abstrahiert. Jeder Partner ordnet einem Geschäftspartner eine Rolle zu.	D
Partners	Sammlung von Partner	D
Pick	Ermöglicht es auf eine bestimmte	-

	Nachricht zu warten (onAlarm, onMessage)	
Process	Markiert Beginn und Ende einer BPEL4WS-Beschreibung	X
Receive	Blockierendes Warten auf Empfang einer Nachricht	-
Reply	Antwort auf empfangene Nachricht schicken (Receive-Reply -> Request-Response-Operation bei PortType)	-
Scope	Bestimmt den Ausführungskontext (Execution Context) von Aktivitäten	-
Sequence	Aktivitäten werden in der textuellen Reihenfolge ausgeführt in der sie im Sequence-Block stehen	X
ServiceLinkType	Assoziiert eine Rolle mit einem PortType (siehe Partner)	D
Switch	if (...) else ...	X
Terminate	Beendet einen SWF	X
Throw	Wirft einen Fehler	-
Wait	Wartet auf einen Termin oder eine gewisse Zeit	-
While	while-Schleife	X

Tabelle 6: BPEL4WS-SWFDL-Vergleichsmatrix

D = Dummy

K = Konzeptuelle Unterstützung

X = Sprachbestandteil

- = kein Sprachbestandteil

13.3 Beispiel für einen Subworkflow

13.3.1 Beschreibung

Dieser Subworkflow beschreibt eine einfache credit-debit-Anwendung, die darin besteht, dass von einer Kreditkarte ein bestimmter Betrag (Abbildung 25-amount) auf ein angegebenes Konto (Abbildung 25-destinationAccountNo) überwiesen wird. Dazu wird anhand der Kreditkartennummer (Abbildung 25-creditCard.number), des Kreditkarteninhabers (Abbildung 25-creditCard.name) und der Gültigkeitsdauer der Kreditkarte (Abbildung 25-creditCard.expirationDate) entschieden, ob die Kreditkarte gültig ist. Sofern dies der Fall ist, erfolgt die Belastung der Kreditkarte (Abbildung 24-credit) (und es werden die dazugehörigen Tracking Informationen geschrieben), gefolgt von der Gutschrift auf das Konto des Empfängers (Abbildung 24-debit) und der SWF endet mit einer Benachrichtigung des Auftraggebers (z.B. über den MFC), dass die Bezahlung erfolgreich war (Abbildung 24-successNotificationActivity). Im Falle, dass die Kreditkarte abgelehnt wird, wird der Auftraggeber (z.B. über den MFC) davon benachrichtigt, dass die Kreditkarte abgelehnt wurde (Abbildung 24-errorNotificationActivity). Außerdem wird die neue Kreditkartenbelastung (Abbildung 25-cardBalance) nach der Überweisung als zusätzlicher Rückgabewert an den Auftraggeber (z.B. über den MFC) zurückgegeben.

Des Weiteren können folgende SWF-spezifische Fehler auftreten:

- Falls die credit-Aktivität nach der Ausführung über den Return Code signalisiert, dass der Fehler RC_ERROR001 aufgetreten ist, werden die globalen und lokalen Daten auf den Stand vor der Ausführung der credit-Aktivität zurückgesetzt und die credit-Aktivität erneut gestartet, da unmittelbar vor der Ausführung der credit-Aktivität ein Savepoint geschrieben wurde (Abbildung 24-save),
- Falls die debit-Aktivität nach der Ausführung über den Return Code signalisiert, dass der Fehler RC_ERROR002 aufgetreten ist, werden die globalen und lokalen Daten auf den Stand zu Beginn der Ausführung des SWF zurückgesetzt und die checkCreditCard-Aktivität erneut gestartet, da unmittelbar vor dieser Aktivität die aktuelle Transaktion gestartet wurde (Abbildung 24 - beginTx).

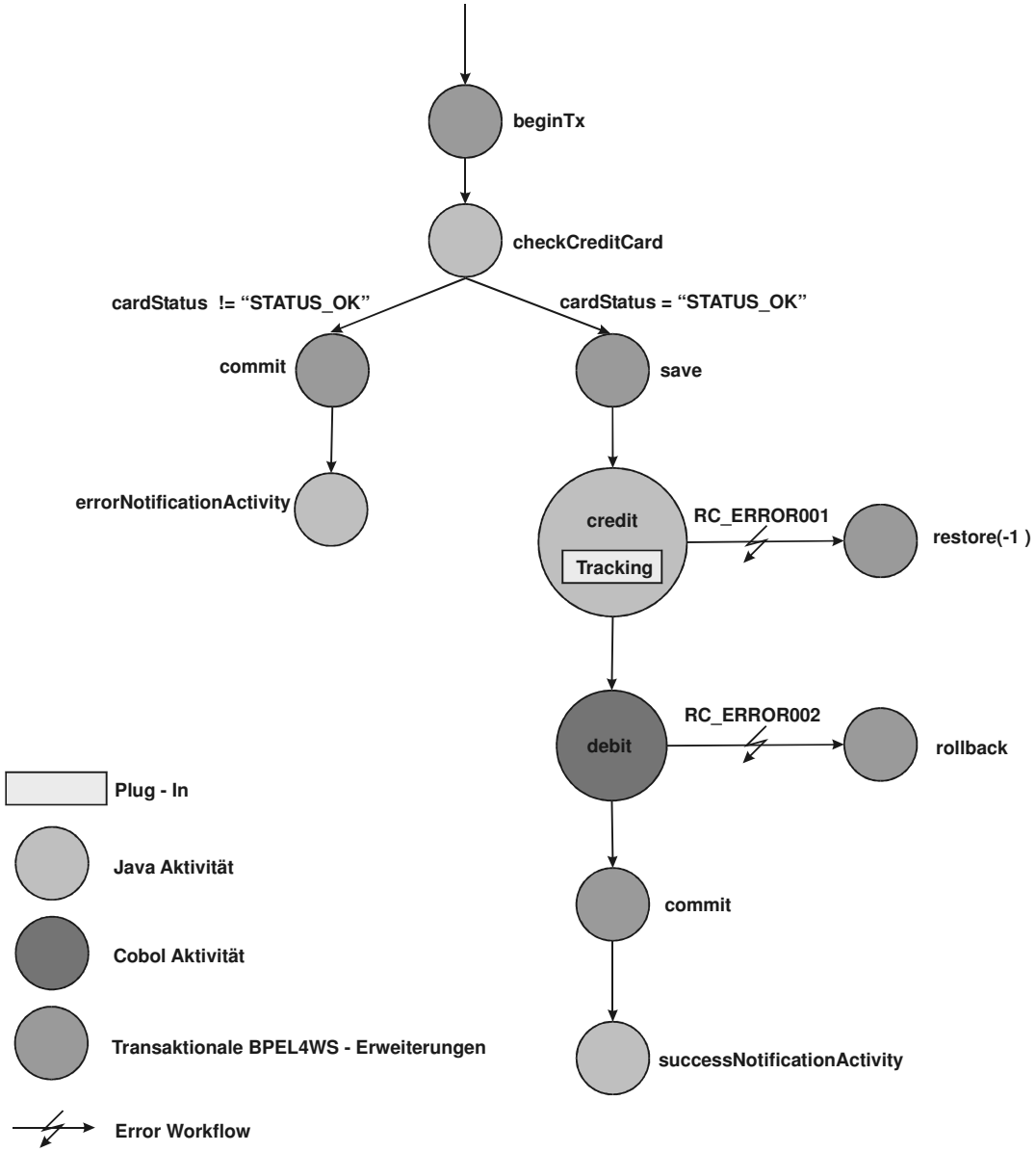

Abbildung 24: Struktur des Beispiel SWF Nr. 23

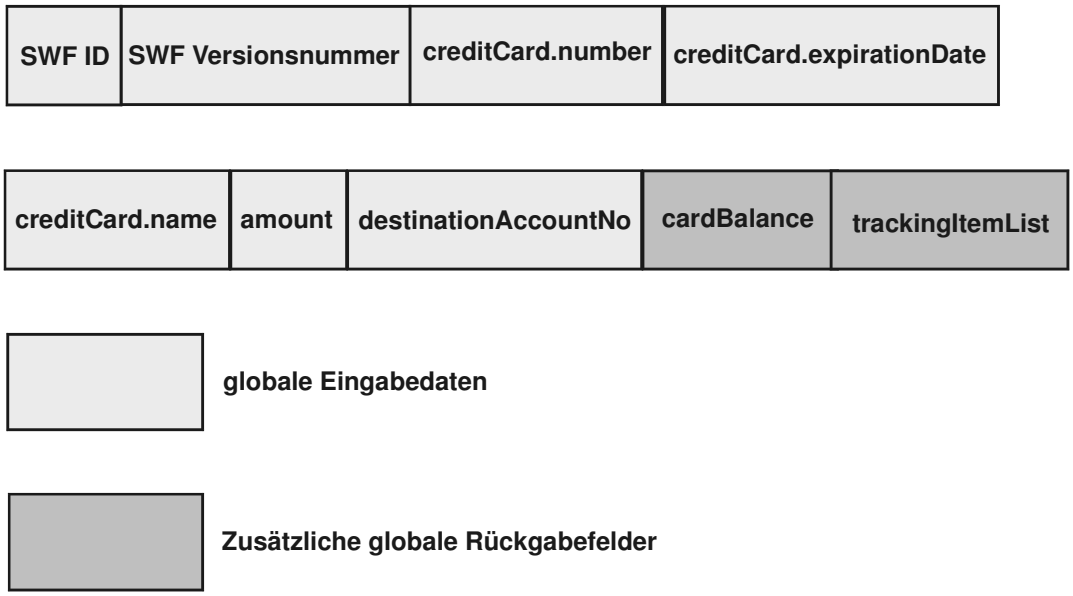

Abbildung 25: Struktur der SWFC RWU (und somit globalen Daten) für den SWF Nr. 23

Abbildung 26: Struktur der lokalen Daten für den SWF Nr. 23

13.3.2WSDL Beschreibung des Subworkflows

Dies ist eine Beispiel WSDL Beschreibung für den in Kapitel 13.3.1 beschriebenen SWF Nr. 23. Hier wird nur eine Input Message und eine Output Message definiert, welche angeben, was der SWFC als Eingabe für den SWF benötigt, bzw. welches Format die Ausgabe hat. Die durch die in Abbildung 24 dargestellten Notification Aktivitäten (successNotificationActivity und errorNotificationActivity) gesendeten Nachrichten werden als Output Message (im Erfolgsfall) bzw. als Error Message der WSDL Operation umgesetzt (Kreditkarte wird zurückgewiesen). Wie in Abbildung 25 dargestellt, muss zusätzlich die ID und Versionsnummer an den SWFC übergeben werden.

Die Umsetzung der Eingabe bzw. der Ausgabe auf das neutrale interne SWFC RWU Format übernimmt an dieser Stelle der I / O Controller.

```
<wsdl:definitions
     name="SWF23"
     targetNamespace="http://www.thornung.com/2003/SWFDL/swf23">
  <wsdl:types>
```

```
    <xsd:schema>
      <xsd:complexType name=”creditCard”>
        <xsd:element name=”number” type=”xsd:string” />
        <xsd:element name=”name” type=”xsd:integer” />
        <xsd:element name=”expirationDate” type=”xsd:gYearMonth” />
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name=“Input“>
    <wsdl:part name=“SWF_ID“ type=“xsd:string“>
    <wsdl:part name=“Version_No“ type=“xsd:string“>
    <wsdl:part name=”cardInformation” type=”creditCard” />
    <wsdl:part name=”amount” type=”xsd:double” />
    <wsdl:part name=”destinationAccountNo” type=”xsd:integer” />
  </wsdl:message>
  <wsdl:message name=“Output“>
    <wsdl:part name=“success“ type=“xsd:string“ />
  </wsdl:message>
  <wsdl:message name=“Fault“>
    <wsdl:part name=“error“ type=“xsd:string“ />
  </wsdl:message>

  <wsdl:portType name=“SWF23PortType“>
    <wsdl:operation name=“creditCardPayment“>
      <wsdl:input message=“Input“>
      </wsdl:input>
      <wsdl:output message=“Output“>
      </wsdl:output>
      <wsdl:fault message=“Fault“>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:serviceType name=“SWF23ST“>
    <wsdl:portType name=“SWF23PortType“ />
```

```
   </wsdl:serviceType>

   <wsdl:binding name="SWF23Binding">
      <wsdl:operation name="creditCardPayment">
         <wsdl:input>
            <!-- binding details -->
         </wsdl:input>
         <wsdl:output>
            <!-- binding details -->
         </wsdl:output>
         <wsdl:fault message="Fault">
            <!-- binding details -->
         </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="SWF23Service" serviceType="SWF23ST">
      <wsdl:port name="SWF23Port" binding="SWF23Binding">
         <!-- address details -->
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

13.3.3SWFDL-Beschreibung des Subworkflows

Dies ist eine Beispiel SWFDL Beschreibung für den in Kapitel 13.3.1 dargestellten SWF Nr. 23. Die SWFDL-Neuerung zu BPEL4WS sind hier **fett** hervorgehoben.

```
<?xml version="1.0" encoding="ebcdic-cp-us"?>
<process  name="SWF23V100"
          xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-
          process"
          xmlns:xsd="http://w3.org/2001/XMLSchema"
          xmlns:swfdl="http://www.thornung.com/2003/SWFDL"
          xmlns:wsdl="http//schemas.xmlsoap.org/wsdl"
          xmlns:slnk="http//schemas.xmlsoap.org/ws/2002/07/service-
          link/">
   <extension namespace="http//www.thornung.com/2003/SWFDL" />
```

```
<wsdl:types>
  <xsd:schema>
    <xsd:complexType name="creditCard">
      <xsd:element name="number" type="xsd:string" />
      <xsd:element name="name" type="xsd:integer" />
      <xsd:element name="expirationDate" type="xsd:gYearMonth" />
    </xsd:complexType>
    <xsd:simpleType name="cardStatus">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="STATUS_OK" />
        <xsd:enumeration value="STATUS_INVALID_NUMBER" />
        <xsd:enumeration value="STATUS_INVALID_NAME" />
        <xsd:enumeration value="STATUS_EXPIRED" />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="trackingItemListType">
      <xsd:sequence>
        <xsd:element  name="trackingItem" type="xsd:string"
                      minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>

<wsdl:message name="checkCreditCardInput">
  <wsdl:part name="cardBalance" type="xsd:double" />
  <wsdl:part name="creditCardInformation" type="creditCard" />
</wsdl:message>
<wsdl:message name="checkCreditCardOutput">
  <wsdl:part name="cardStatus" type="cardStatus" />
</wsdl:message>
<wsdl:message name="creditInput">
  <wsdl:part name="amount" type="xsd:double" />
```

```
    <wsdl:part name="cardBalance" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="creditOutput">
    <wsdl:part name="amount" type="xsd:double" />
    <wsdl:part name="cardBalance" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="debitInput">
    <wsdl:part name="amount" type="xsd:double" />
    <wsdl:part name="balanceDestinationAccount" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="debitOutput">
    <wsdl:part name="balanceDestinationAccount" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="errorNotificationActivityInput">
    <wsdl:part name="cardStatus" part="cardStatus" />
  </wsdl:message>
  <wsdl:message name="successNotificationActivityInput">
    <wsdl:part name="cardBalance" type="xsd:double" />
    <wsdl:part name="balanceDestinationAccount" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="trackingInput">
    <wsdl:part name="trackingItemList" type="trackingItemListType"/>
  </wsdl:message>
  <wsdl:message name="trackingOutput">
    <wsdl:part name="trackingItemList" type="trackingItemListType"/>
  </wsdl:message>

  <wsdl:portType name="SWF23V100PortType">
    <wsdl:operation name="checkCreditCard">
      <wsdl:input message="checkCreditCardInput">
      </wsdl:input>
      <wsdl:output message="checkCreditCardOutput">
      </wsdl:output>
    </wsdl:operation>
```

```
    <wsdl:operation name="credit">
      <wsdl:input message="creditInput">
      </wsdl:input>
      <wsdl:output message="creditOutput">
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="debit">
      <wsdl:input message="debitInput">
      </wsdl:input>
      <wsdl:output message="debitOutput">
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="errorNotificationActivity">
      <wsdl:input message="errorNotificationActivityInput">
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="successNotificationActivity">
      <wsdl:input message="successNotificationActivityInput">
      </wsdl:input>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="PluginPortType">
    <wsdl:operation name="tracking">
      <wsdl:input message="trackingInput">
      </wsdl:input>
      <wsdl:output message="trackingOutput">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>

  <swfdl:binding name="SWF23V100Binding" type="SWF23V100PortType">
    <swfdl:function operation="checkCreditCard">
      <swfdl:javaOperation  javaMethodName="checkCreditCard"
```

```
                        javaClassName="SWF23V100" />
    </swfdl:function>
    <swfdl:function operation="credit">
      <swfdl:javaOperation  javaMethodName="credit"
                        javaClassName="SWF23V100" />
    </swfdl:function>
    <swfdl:function operation="errorNotificationActivity">
      <swfdl:javaOperation  javaMethodName="error"
                        javaClassName="SWF23V100" />
    </swfdl:function>
    <swfdl:function operation="successNotificationActivity">
      <swfdl:javaOperation  javaMethodName="success"
                        javaClassName="SWF23V100" />
    </swfdl:function>
    <swfdl:function operation="debit">
      <swfdl:cobolOperation   cobolFunctionName="debit"
                          cobolDllName="libSWF23.so" />
    </swfdl:function>
  </swfdl:binding>

  <swfdl:binding name="PluginBinding" type="PluginPortType">
    <swfdl:function operation="tracking">
      <swfdl:javaOperation  javaMethodName="tracking"
                        javaClassName="SWF23V100Plugin" />
    </swfdl:function>
  </swfdl:binding>

  <partners>
    <partner name="dummyPartner" serviceLinkType="dummyLT"
             myRole="dummyRole"/>
  </partners>

  <variables>
    <variable name="counterDataInput" messageType="counterDataInput"/>
```

```
    <variable   name="counterDataOutput"
                messageType="counterDataOutput"/>
    <variable   name="checkCreditCardInput"
                messageType="checkCreditCardInput"/>
    <variable   name="checkCreditCardOutput"
                messageType="checkCreditCardOutput"/>
    <variable name="creditInput" messageType="creditInput"/>
    <variable name="creditOutput" messageType="creditOutput"/>
    <variable name="debitInput" messageType="debitInput"/>
    <variable name="debitOutput" messageType="debitOutput"/>
    <variable   name="errorNotificationActivityInput"
                messageType="errorNotificationActivityInput"/>
    <variable   name="successNotificationActivityInput"
                messageType="successNotficationActivityInput"/>
  </variables>

  <sequence>
    <swfdl:beginTx />

    <invoke   name="checkCreditCard"
              partner="dummyPartner"
              portType="SWF23V100PortType"
              operation="checkCreditCard"
              inputVariable="checkCreditCardInput"
              outputVariable="checkCreditCardOutput">
    </invoke>

    <switch>
      <case condition= "bpws:getVariableData( checkCreditCardOutput,
                      cardStatus ) != 'STATUS_OK'">
        <sequence>
          <swfdl:commitTx />

          <invoke   name="errorNotificationActivity"
```

```
                partner="dummyPartner"
                portType="SWF23V100PortType"
                operation="errorNotificationActivity"
                inputVariable="errorNotificationActivityInput">
      </invoke>
    </sequence>
  </case>
  <case condition= "bpws:getVariableData( checkCreditCardOutput,
                  cardStatus ) = 'STATUS_OK'">
    <sequence>
      <swfdl:saveTx />

      <invoke   name="credit"
                partner="dummyPartner"
                portType="SWF23V100PortType"
                operation="credit"
                inputVariable="creditInput"
                outputVariable="creditOutput">
        <catch faultName="RC_ERROR001">
          <swfdl:restoreTx whichSavepoint="0" />
        </catch>

        <swfdl:plugin operation="tracking" />
      </invoke>
      <invoke   name="debit"
                partner="dummyPartner"
                portType="SWF23V100PortType"
                operation="debit"
                inputVariable="debitInput"
                outputVariable="debitOutput">
        <catch  faultName="RC_ERROR002">
          <swfdl:rollbackTx />
        </catch>
      </invoke>
```

```
            <swfdl:commitTx />

            <invoke   name="successNotificationActivity"
                      partner="dummyPartner"
                      portType="SWF23V100PortType"
                      operation="successNotificationActivity"
                      inputVariable=
                      "successNotificationActivityInput">
            </invoke>
          </sequence>
        </case>
      </switch>
    </sequence>
</process>
```

13.4 Beschreibung der Finite State Machine (FSM)

Die Ausführung der FSM verläuft wie in den folgenden zwei Tabellen dargestellt und in dem State Diagram (Abbildung 27) dargestellt:

Ereignis #	Name des Verarbeitungsschritts	Verarbeitungsschritt(e)
1	Start SWFC	Der Carrier startet den SWFC, der anschließend (sofern vorhanden) eine Konfigurationsdatei einliest und parst. Anschließend startet der SWFC alle übrigen Module. Danach prüft er, ob noch eine alte (nicht bearbeitete) SWFC RWU gespeichert ist, bzw. ob noch ein alter Savepoint existiert und führt diese sofern vorhanden aus, bzw. setzt den SWF an der Stelle des Savepoints mit einem entsprechenden außerordentlichen Zustandswechsel (dies ermöglicht es, den SWF im richtigen Zustand wieder fortzusetzen) fort. Anschließend startet der SWFC die FSM und die Endlos-Schleife, in der er über den I / O Controller auf ankommende Aufträge zur Verarbeitung wartet.
2	Initiate SWF	Der I / O Controller liest ein SWFC RWU von der

		In Queue und es wird eine SWF Repräsentation und ein Message Container erzeugt, die an den SWF Interpreter übergeben werden.
3	Initiate SWF Activity	Der SWF Interpreter erstellt bei einer Java SWF Aktivität eine lokale Sicht des Message Containers und bei einer COBOL SWF Aktivität ein Java COBOL Objekt.
4	Execute SWF Activity, parallel – Not last	Wie Execute SWF Activity, mit dem Unterschied, dass dies nicht die letzte Aktivität im Zustand Initiated SWF Activity ist.
5	Execute SWF Activity, (parallel – last)	Der SWF Interpreter führt die Aktivität aus.
6	Release SWF Activity, RC_OK & RC_ERROR parallel – Not last	Wie Release SWF Activity, RC_OK & RC_ERROR, mit dem Unterschied, dass dies nicht die letzte Aktivität im Zustand Executing SWF Activity ist.
7	Release SWF Activity, RC_RECOVER	Der SWF Interpreter startet die Aktivität erneut mit denselben Eingabedaten und informiert den MFC über das Notification API davon, dass die Aktivität erneut gestartet wird.
8	Release SWF Activity, RC_ABORT	Der SWF Interpreter bricht die Ausführung des Subworkflows ab und informiert den MFC über das Notification API darüber, dass die Ausführung des Subworkflows abgebrochen wurde.
9	Abort finished	Mit diesem Ereignis ist kein Verarbeitungsschritt verbunden.
10	Release SWF Activity, RC_ROLLBACKTX	Der Message Container setzt die globalen und lokalen Daten auf den Stand des letzten commitTx bzw. beginTx zurück und liefert als Resultat die Statusinformationen für den SWF Interpreter, der den MFC über das Notification API davon informiert, dass ein rollbackTx stattfindet.
11	RC_ROLLBACKTX, parallel	Keiner. Der SWFC ignoriert alle Ereignisse im Zustand Rollbacking SWF Activity, bis auf SWFC_EVENT_ROLLBACKTX_COMPLETED
12	Rollback completed	Mit diesem Ereignis ist kein Verarbeitungsschritt verbunden.
13	Release SWF Activity,	Der Message Container setzt die globalen und

	RC_RESTORE	lokalen Daten auf den Stand beim letzten Savepoint zurück und liefert als Resultat die Statusinformationen für den SWF Interpreter, der den MFC über das Notification API davon informiert, dass der SWF auf den letzten Savepoint zurückgesetzt wird.
14	RC_RESTORE, parallel	Mit diesem Ereignis ist kein Verarbeitungsschritt verbunden. Der SWFC ignoriert alle Ereignisse im Zustand Restoring SWF Activity, bis auf SWFC_EVENT_RESTORE_COMPLETED
15	Restore completed	Mit diesem Ereignis ist kein Verarbeitungsschritt verbunden.
16	Release SWF Activity, RC_OK & RC_ERROR (parallel – last)	• RC_OK: Der SWF Interpreter synchronisiert bei einer Java SWF Aktivität die eventuell geänderten Daten in der lokalen Sicht und bei einer COBOL SWF Aktivität die eventuell geänderten Daten im Java COBOL Objekt wieder mit den Daten im Message Container und führt anschließend die für diese Aktivität registrierten Plugins aus. • RC_ERROR: Die normale Ausführung des Subworkflows wird abgebrochen und der SWF Interpreter setzt die Ausführung mit dem entsprechenden Error Workflow fort und informiert den MFC über das Notification API davon, dass der Fehler xxx aufgetreten ist.
17	Next Activity	Der SWF Interpreter setzt die Ausführung des Subworkflows mit der nächsten Aktivität fort.
18	SWF finished	Mit diesem Ereignis ist kein Verarbeitungsschritt verbunden.
19	Release SWF	Nach erfolgreicher Beendigung des Subworkflows wird aus dem Message Container ein Java RWU Objekt erzeugt, welches dann wieder an den SWFC als Resultat des fehlerfreien Subworkflows zurückgegeben wird. Dieser gibt das Java RWU Objekt an den I / O Controller weiter, die dieses wieder in das entsprechende Datenpufferformat für die Out Queue überführt und das

		Resultat (die SWFC RWU) für den MFC in die Out Queue legt. Anschließend signalisiert der I / O Controller dem Data Service, das der letzte Savepoint gelöscht werden soll, damit im Fall eines Systemcrash der SWF nicht erneut ausgeführt wird.
20	Found Shutdown SWFC RWU	Der SWFC informiert den MFC über das Notification API davon, dass er sich beendet.
21	End SWFC	Der SWFC beendet die FSM und anschließend informiert er alle Module davon, dass sie sich beenden sollen (z.B. über einen Funktionsaufruf) und beendet sich danach selbst.

Tabelle 7: Liste der Ereignisse, Teil 1

#	Name des Ereignis	Gesendet von
1	SWFC_EVENT_STARTUP_SWFC	SWFC
2	SWFC_EVENT_INITIATE_SWF	SWF Interpreter
3	SWFC_EVENT_INITIATE_SWF_ACTIVITY	SWF Interpreter
4	SWFC_EVENT_EXECUTE_SWF_ACTIVITY_PARALLEL_NOT_LAST	SWF Interpreter
5	SWFC_EVENT_EXECUTE_SWF_ACTIVITY bzw. SWFC_EVENT_EXECUTE_SWF_ACTIVITY_PARALLEL_LAST	SWF Interpreter
6	SWFC_EVENT_RC_OK_PARALLEL_NOT_LAST bzw. SWFC_EVENT_RC_ERROR_PARALLEL_NOT_LAST	SWF Interpreter
7	SWFC_EVENT_RC_RECOVER	SWF Interpreter
8	SWFC_EVENT_RC_ABORT	SWF Interpreter
9	SWFC_EVENT_ABORT_COMPLETED	SWF Interpreter
10	SWFC_EVENT_RC_ROLLBACKTX	SWF Interpreter

11	Alle Ereignisse außer SWFC_EVENT_ROLLBACKTX_COMPLETED	SWF Interpreter
12	SWFC_EVENT_ROLLBACKTX_COMPLETED	SWF Interpreter
13	SWFC_EVENT_RC_RESTORE	SWF Interpreter
14	Alle Ereignisse außer SWFC_EVENT_RESTORE_COMPLETED	SWF Interpreter
15	SWFC_EVENT_RESTORE_COMPLETED	SWF Interpreter
16	SWFC_EVENT_RC_OK bzw. SWFC_EVENT_RC_ERROR bzw. SWFC_EVENT_RC_OK_PARALLEL_LAST bzw. SWFC_EVENT_RC_ERROR_PARALLEL_LAST	SWF Interpreter
17	SWFC_EVENT_NEXT_ACTIVITY	SWF Interpreter
18	SWFC_EVENT_SWF_FINISHED	SWF Interpreter
19	SWFC_EVENT_RELEASE_SWF	SWF Interpreter
20	SWFC_EVENT_FOUND_SHUTDOWN_RWU	SWFC
21	SWFC_EVENT_SHUTDOWN_SWFC	SWFC

Tabelle 8: Liste der Ereignisse, Teil 2

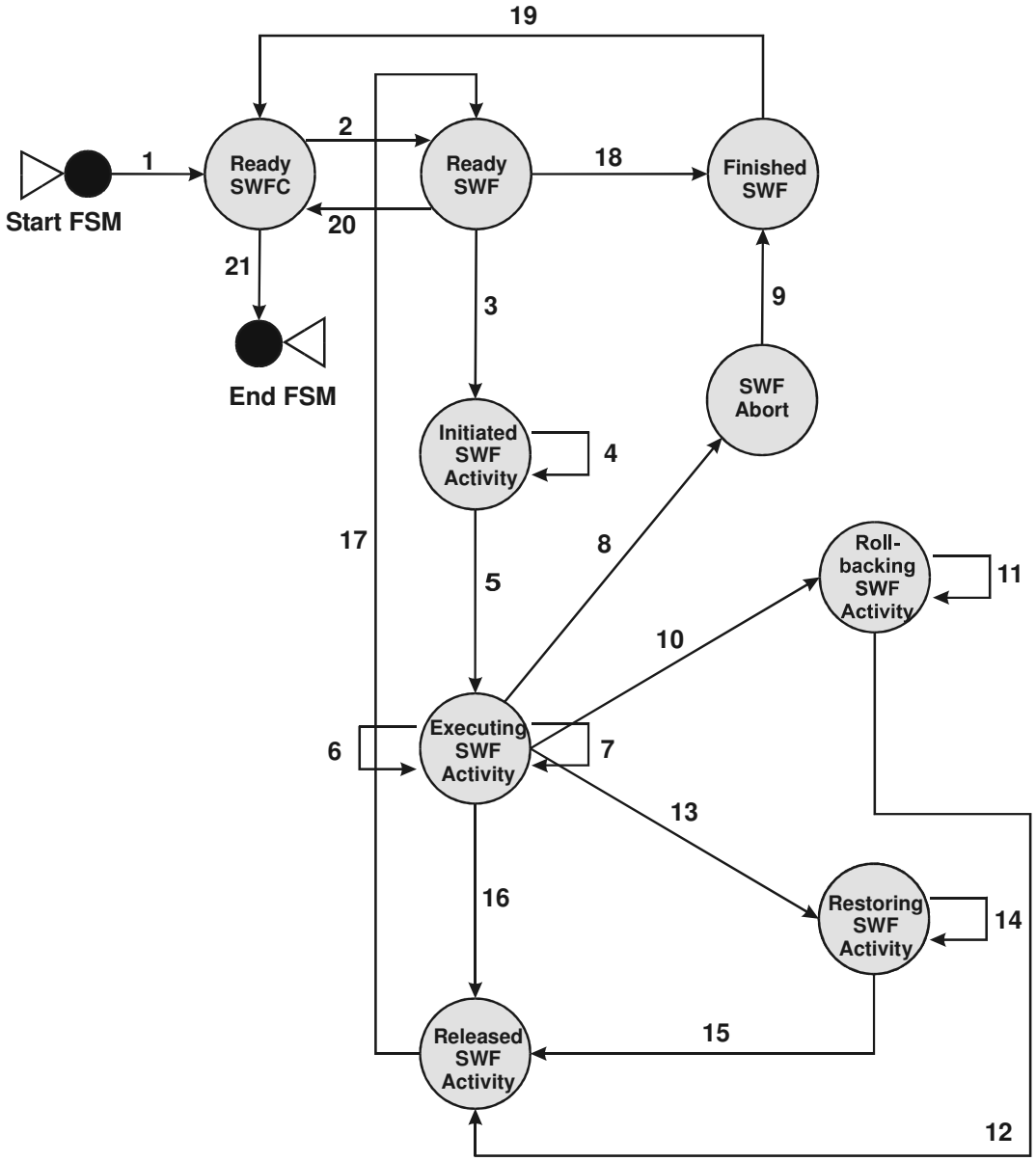

Abbildung 27: State Diagram des SWFC

13.5 Liste der Return Codes

Return Code	Beschreibung
RC_OK	Die Aktivität wurde erfolgreich ausgeführt.
RC_RECOVER	Die Aktivität wurde nicht erfolgreich ausgeführt, kann aber mit den gleichen Eingabewerten erneut gestartet werden.
RC_RESTORE	Die Aktivität wurde nicht erfolgreich ausgeführt und es muss auf den letzten Savepoint zurückgesetzt werden.
RC_ROLLBACKTX	Die Aktivität wurde nicht erfolgreich ausgeführt und es muss auf

	den letzten beginTx- bzw. commitTx-Punkt zurückgesetzt werden.
RC_ABORT	Die Aktivität wurde nicht erfolgreich ausgeführt und es muss die Ausführung des gesamten SWF abgebrochen werden.
RC_ERRORxxx	Die Aktivität wurde nicht erfolgreich ausgeführt und es trat der benutzerdefinierte Fehler xxx, wobei xxx eine beliebige Nummer ist (z.B. 42), auf. Die Fehlerbehandlung erfolgt in einem separaten benutzerdefinierten Error Workflow.
RC_FAILED	Die Aktivität wurde nicht erfolgreich ausgeführt, die Ausführung des SWF wird aber trotzdem fortgesetzt (dieser Return Code kann nur von einer Plugin Aktivität zurückgegeben werden!).

Tabelle 9: Liste der Return Codes

13.6 Use Cases

13.6.1 Überblick

Aus Gründen der Übersichtlichkeit wurde davon abgesehen in diesen Use Cases das Senden von Ereignissen explizit zu beschreiben. Dies ist in Kapitel 13.4: Beschreibung der Finite State Machine (FSM) beschrieben und es wird in diesem Kapitel vorausgesetzt, dass dem SWFC die jeweiligen entsprechenden Ereignisse gesendet werden und die dazugehörigen Verarbeitungsschritte nur ausgeführt werden, wenn sie zu legalen Zustandswechseln führen, bzw. wenn sie in diesem Zustand legal sind.

Generell gilt, dass alle Use Cases den in Kapitel 13.3: Beispiel für einen Subworkflow beschriebenen Subworkflow nutzen.

13.6.2 Use Case: Starten und Beenden des SWFC

13.6.2.1 Szenario: Starten des SWFC – der SWFC wurde normal beendet

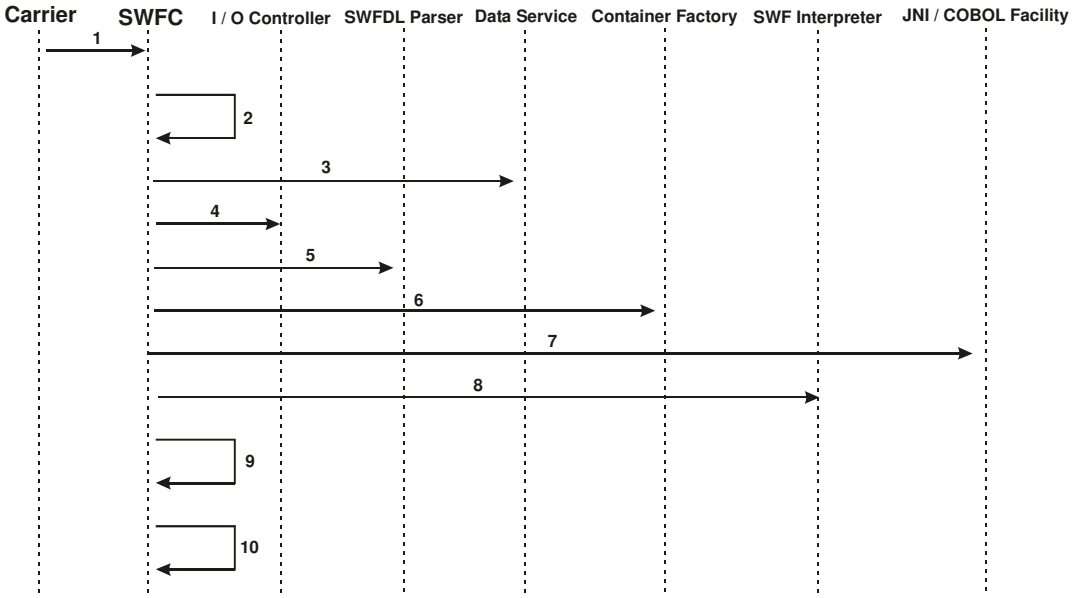

Abbildung 28: Use Case #01: Starten des SWFC – der SWFC wurde normal beendet

Use Case #01	**Starten und Beenden des SWFC**
Szenario Name	Starten des SWFC – der SWFC wurde normal beendet
Benutzter SWF	Keiner
Use Case Übersicht	Dieser Use Case beschreibt alle relevanten Szenarien die beim Starten und Beenden des SWFC auftreten können.
Zustand Beginn	Undefiniert
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWFC ist installiert und der Carrier ist in der Lage ihn zu starten (z.B. über einen Funktionsaufruf).
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	Keine
SWFC RWU nach Szenario	Keine
Lokale Daten vor Szenario	Keine

Lokale Daten nach Szenario	Keine
Use Case Beschreibung	
Der Carrier startet den SWFC, der anschließend die FSM startet und (sofern vorhanden) eine Konfigurationsdatei einliest und parst (Abbildung 28-1 und Abbildung 28-2). Anschließend startet der SWFC alle übrigen Module (Abbildung 28-3 bis Abbildung 28-8). Danach prüft er, ob noch ein alte (nicht bearbeitete) SWFC RWU gespeichert ist, bzw. ob noch ein alter Savepoint existiert (Abbildung 28-9 bis Abbildung 28-10). Dies ist nicht der Fall und somit startet der SWFC die Endlos-Schleife, in der er über den I / O Controller auf ankommende Aufträge zur Verarbeitung wartet.	
Verwandte Use Cases	Use Case #02, Use Case #03, Use Case #04
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Alte (nicht bearbeitete) SWFC RWU	Nicht vorhanden, somit keine Auswirkung
Alter Savepoint	Nicht vorhanden, somit keine Auswirkung

Tabelle 10: Use Case #01: Starten des SWFC – der SWFC wurde normal beendet

13.6.2.2 Szenario: Starten des SWFC – der SWFC wurde nicht normal beendet, eine nicht bearbeitete SWFC RWU ist vorhanden

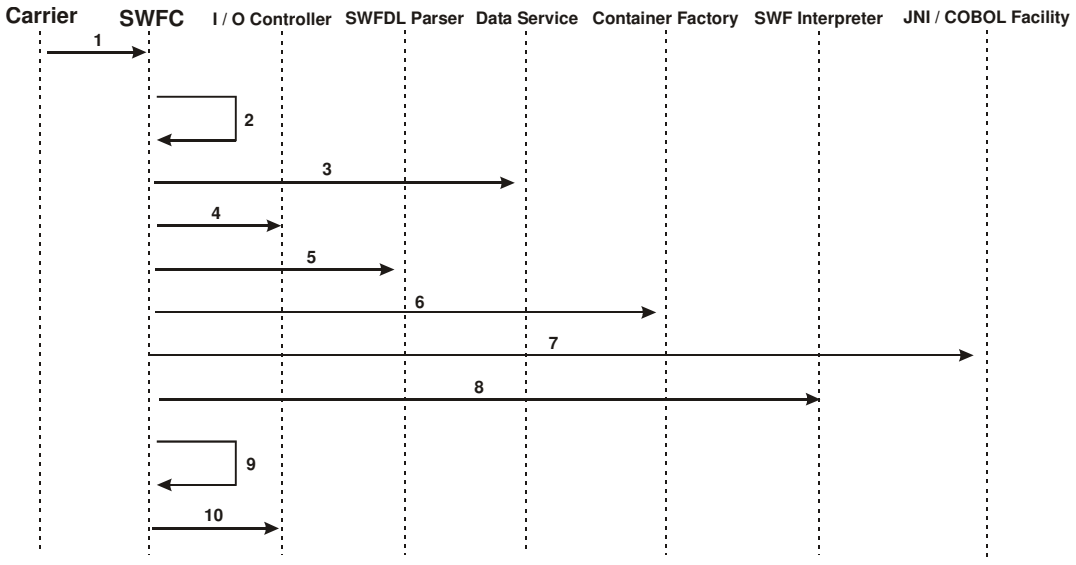

Abbildung 29: Use Case #02: Starten des SWFC – der SWFC wurde nicht normal beendet, eine nicht bearbeitete SWFC RWU ist vorhanden

Use Case #02	**Starten und Beenden des SWFC**
Szenario Name	Starten des SWFC – der SWFC wurde nicht normal beendet, eine nicht bearbeitete SWFC RWU ist vorhanden
Benutzter SWF	Keiner
Use Case Übersicht	Dieser Use Case beschreibt alle relevanten Szenarien die beim Starten und Beenden des SWFC auftreten können.
Zustand Beginn	Undefiniert
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWFC ist installiert und der Carrier ist in der Lage ihn zu starten (z.B. über einen Funktionsaufruf).
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	Keine
SWFC RWU nach Szenario	Keine
Lokale Daten vor Szenario	Keine
Lokale Daten nach Szenario	Keine
Use Case Beschreibung	
Der Carrier startet den SWFC, der als die FSM startet und (sofern vorhanden) eine Konfigurationsdatei einliest und parst (Abbildung 29-1 und Abbildung 29-2). Anschließend startet der SWFC alle übrigen Module (Abbildung 29-3 bis Abbildung 29-8). Danach prüft er, ob noch ein alte (nicht bearbeitete) SWFC RWU gespeichert ist (Abbildung 29-9). Dies ist der Fall und der SWFC übergibt die SWFC RWU an den I / O Controller (Abbildung 29-10), der daraus ein Java RWU Objekt erzeugt. Daraufhin wird, wie in den Use Cases #05 bzw. #10 beschrieben, der Auftrag bearbeitet. Nachdem die bearbeitete SWFC RWU wieder in die Out Queue gelegt wurde, startet der SWFC die Endlos-Schleife, in der er über den I / O Controller auf ankommende Aufträge zur Verarbeitung wartet.	
Verwandte Use Cases	Use Case #01, Use Case #03, Use Case #04, Use Case #05, Use Case #10
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Alte (nicht bearbeitete) SWFC RWU	SWFC RWU wird über I / O Controller in ein Java RWU Objekt überführt und anschließend verarbeitet.

Alter Savepoint	Nicht vorhanden, somit keine Auswirkung

Tabelle 11: Use Case #02: Starten des SWFC – der SWFC wurde nicht normal beendet, eine nicht bearbeitete SWFC RWU ist vorhanden

13.6.2.3 Szenario: Starten des SWFC – der SWFC wurde nicht normal beendet, eine alter Savepoint ist vorhanden

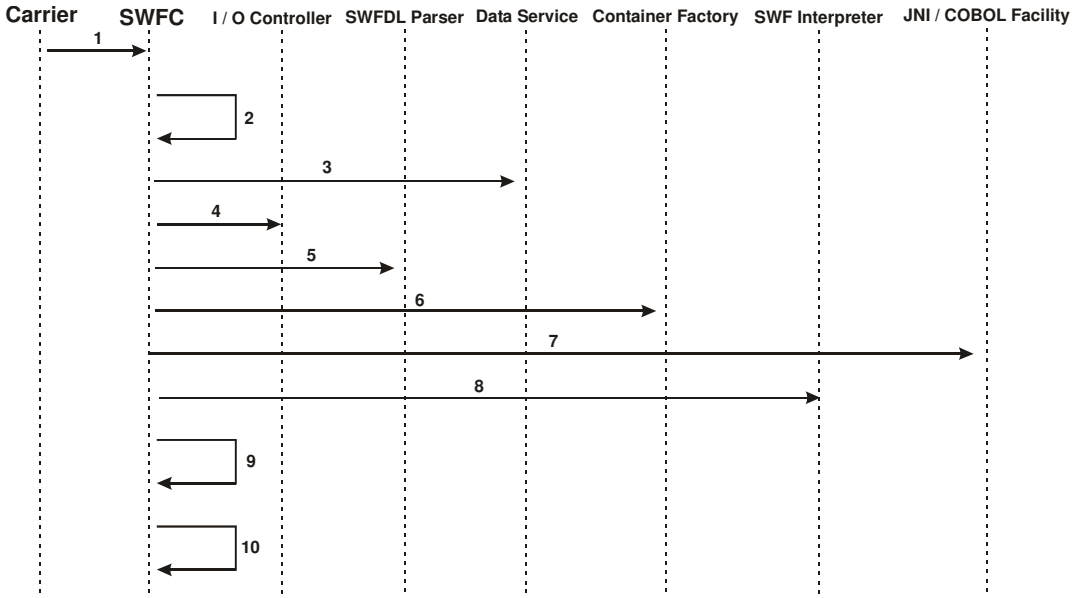

Abbildung 30: Use Case #03: Starten des SWFC – der SWFC wurde nicht normal beendet, ein alter Savepoint ist vorhanden

Use Case #03	**Starten und Beenden des SWFC**
Szenario Name	Starten des SWFC – der SWFC wurde nicht normal beendet, eine alter Savepoint ist vorhanden
Benutzter SWF	Keiner
Use Case Übersicht	Dieser Use Case beschreibt alle relevanten Szenarien die beim Starten und Beenden des SWFC auftreten können.
Zustand Beginn	Undefiniert
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWFC ist installiert und der Carrier ist in der Lage ihn zu starten (z.B. über einen Funktionsaufruf).
Eingabedaten von MFC	Keine

Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	Keine
SWFC RWU nach Szenario	Keine
Lokale Daten vor Szenario	Keine
Lokale Daten nach Szenario	Keine
Use Case Beschreibung	
Der Carrier startet den SWFC, der anschließend die FSM startet und (sofern vorhanden) eine Konfigurationsdatei einliest und parst (Abbildung 30-1 und Abbildung 30-2). Anschließend startet der SWFC alle übrigen Module (Abbildung 30-3 bis Abbildung 30-8). Danach prüft er, ob noch ein alte (nicht bearbeitete) SWFC RWU gespeichert ist, was nicht der Fall ist (Abbildung 30-9). Jedoch existiert noch ein alter Savepoint (Abbildung 30-10). Daraufhin setzt er wie in Use Case #19 beschrieben die Ausführung des Subworkflows an der entsprechenden Stelle fort und nachdem die bearbeitete SWFC RWU in die Out Queue gelegt wurde startet der SWFC die Endlos-Schleife, in der er über den I / O Controller auf ankommende Aufträge zur Verarbeitung wartet.	
Verwandte Use Cases	Use Case #01, Use Case #02, Use Case #04, Use Case #18
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Alte (nicht bearbeitete) SWFC RWU	Nicht vorhanden, somit keine Auswirkung
Alter Savepoint	Die Ausführung des SWF wird an der entsprechenden Stelle wieder fortgesetzt.

Tabelle 12: Use Case #03: Starten des SWFC – der SWFC wurde nicht normal beendet, ein alter Savepoint ist vorhanden

13.6.2.4 Szenario: Beenden des SWFC

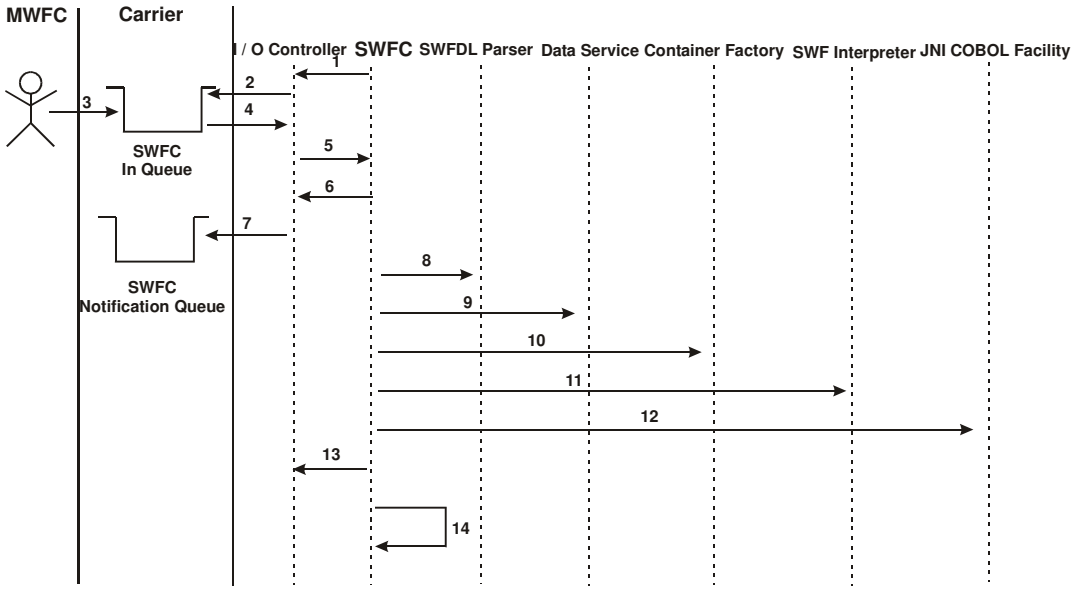

Abbildung 31: Use Case #04: Beenden des SWFC

Use Case #04	**Starten und Beenden des SWFC**
Szenario Name	Beenden des SWFC
Benutzter SWF	Keiner
Use Case Übersicht	Dieser Use Case beschreibt alle relevanten Szenarien die beim Starten und Beenden des SWFC auftreten können.
Zustand Beginn	Ready SWFC
Zustand Ende	Undefiniert
Vorbedingungen	Der SWFC ist bereit hat gerade eine bearbeitete SWFC RWU in die Out Queue gelegt und die nächste SWFC RWU in der In Queue ist die SWFC SHUTDOWN RWU
Eingabedaten von MFC	SWFC SHUTDOWN RWU: • 000 (SWF ID) • 000 (SWF Versionsnummer)
Rückgabedaten an MFC	Notification Message an den MFC (z.B. SHUTING DOWN SWFC)
SWFC RWU vor Szenario	Keine
SWFC RWU nach Szenario	Keine

Lokale Daten vor Szenario	Keine
Lokale Daten nach Szenario	Keine
Use Case Beschreibung	
Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs). Wenn der MFC die SWFC SHUTDOWN RWU in die In Queue des SWFC legt, beendet der SWFC die FSM (Abbildung 31-1 bis Abbildung 31-5) und informiert anschließend den MFC über das Notification API davon, dass er sich beendet (Abbildung 31-6 bis Abbildung 31-7). Anschließend informiert er alle Module davon, dass sie sich beenden sollen (z.B. über einen Funktionsaufruf) und beendet sich danach selbst (Abbildung 31-8 bis Abbildung 31-14).	
Verwandte Use Cases	Use Case #01, Use Case #02, Use Case #03
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Keine	Keine

Tabelle 13: Use Case #04: Beenden des SWFC

13.6.3 Use Case: Initialisierung eines Subworkflows

13.6.3.1 Szenario: Gültige SWFC RWU, erster Start des Subworkflows

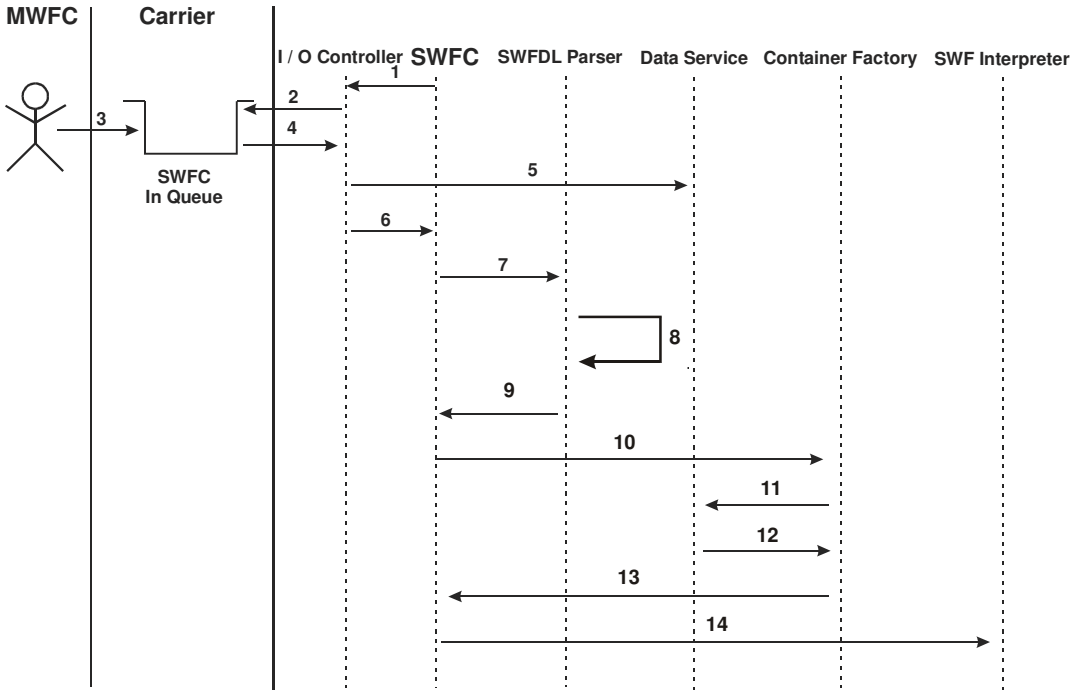

Abbildung 32: Use Case #05: Gültige SWFC RWU, erster Start des Subworkflows

Use Case #05	**Initialisierung eines Subworkflows**
Szenario Name	Gültige SWFC RWU, erster Start des Subworkflows
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die erforderlichen Initialisierungen, die für die Ausführung eines Subworkflows benötigt werden.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWF
Vorbedingungen	Der SWFC ist bereit und verfügbar.
Eingabedaten von MFC	SWFC RWU: • 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • empty (cardBalance) • empty (trackingItemList)
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)

Lokale Daten vor Szenario	Lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	• empty (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs) (Abbildung 32-1 und Abbildung 32-2). Wenn der MFC eine SWFC RWU in die In Queue des SWFC legt, liest der I / O Controller diese von der Queue, persistiert sie und „liest“ anhand der SWF ID und der SWF Versionsnummer die entsprechende Data Section Copystruktur und Data Description (über Reflection in Java) ein (Abbildung 32-3 bis Abbildung 32-4). Diese speichert der I / O Controller für eine spätere Ausführung dieses Subworkflows zwischen. Anschließend gibt der I / O Controller die Data Description weiter an den Data Service, der diese für den Zeitraum der Ausführung dieses Subworkflows zwischenspeichert (Abbildung 32-5). Daraufhin erzeugt der I / O Controller aus der SWFC RWU mit Hilfe der Data Section Copystruktur ein Java RWU Objekt und übergibt diese an den SWFC (Abbildung 32-6). Dieser gibt dann die SWF ID und SWF Versionsnummer weiter an den SWFDL Parser, der die entsprechende SWFDL-Beschreibung einliest, parst und daraus eine SWF Repräsentation erstellt, die er an den SWFC zurückgibt, der diese zwischenspeichert (Abbildung 32-7 bis Abbildung 32-9). Anschließend gibt der SWFC das Java RWU Objekt an die Container Factory, welche die fehlenden Felder mit Hilfe des Data Service füllt und daraus den Message Container initialisiert und an den SWFC zurückgibt (Abbildung 32-11 bis Abbildung 32-14). Zum Schluss wird der Message Container und die SWF Repräsentation an den SWF Interpreter übergeben (Abbildung 32-15).	
Verwandte Use Cases	Use Case #02, Use Case #06, Use Case #07, Use Case #08, Use Case #09
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Gültige, nicht korrupte SWFC RWU	Die SWF Repräsentation wird generiert und der Message Container wird initialisiert und beide werden an den SWF Interpreter übergeben.

Tabelle 14: Use Case #05: Initialisierung eines Subworkflows – Gültige SWFC RWU, erster Start des Subworkflows

13.6.3.2 Szenario: Gültige SWFC RWU, zweiter Start des Subworkflows

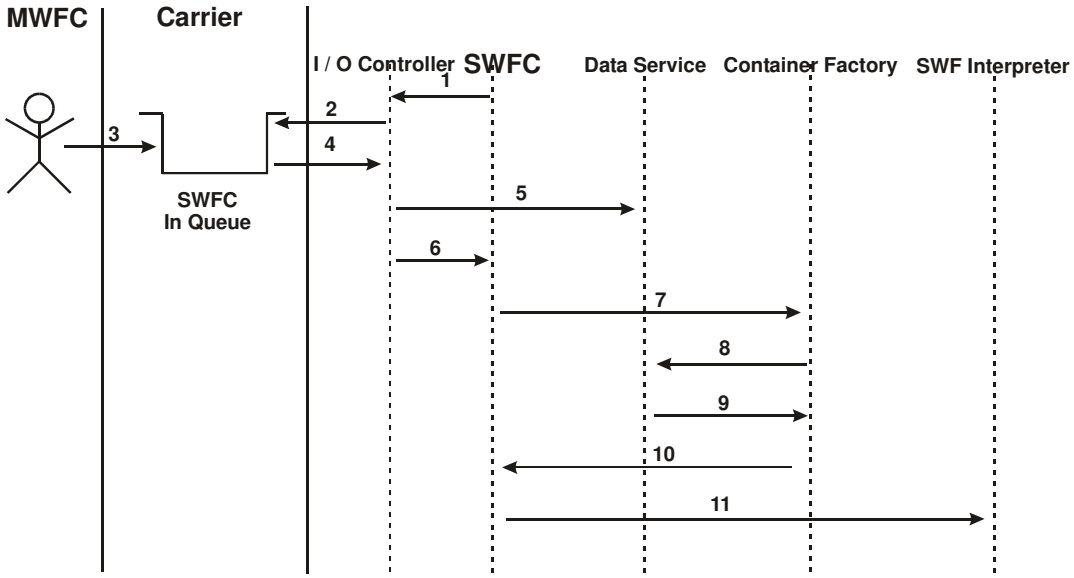

Abbildung 33: Use Case #06: Gültige SWFC RWU, zweiter Start des Subworkflows

Use Case #06	**Initialisierung eines Subworkflows**
Szenario Name	Gültige SWFC RWU, zweiter Start des Subworkflows
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die erforderlichen Initialisierungen, die für die Ausführung eines Subworkflows benötigt werden.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWF
Vorbedingungen	• Der SWFC ist bereit und verfügbar. • Die Data Description und die Data Section Copystruktur ist bereits vom I / O Controller zwischengespeichert. • Die SWF Repräsentation ist bereits vom SWFC zwischengespeichert.
Eingabedaten von MFC	SWFC RWU: • 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number)

	• 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • empty (cardBalance) • empty (trackingItemList)
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	Lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	• empty (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	

Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs) (Abbildung 33-1 und Abbildung 33-2). Wenn der MFC eine SWFC RWU in die In Queue des SWFC legt, liest der I / O Controller diese von der Queue (Abbildung 33 – 3 bis Abbildung 33-4) und persistiert sie. Anschließend gibt der I / O Controller die entsprechende zwischengespeicherte Data Description weiter an den Data Service, der diese für den Zeitraum der Ausführung dieses Subworkflows zwischenspeichert (Abbildung 33-5). Daraufhin erzeugt der I / O Controller aus der SWFC RWU mit Hilfe der entsprechenden zwischengespeicherten Data Section Copystruktur ein Java RWU Objekt und übergibt dieses an den SWFC (Abbildung 33-6). Dieser gibt das Java RWU Objekt an die Container Factory, welche die fehlenden Felder mit Hilfe des Data Service füllt und damit den Message Container initialisiert und an den SWFC zurückgibt (Abbildung 33-7 bis Abbildung 33-10). Zum Schluss wird der Message Container und die entsprechende zwischengespeicherte SWF Repräsentation an den SWF Interpreter übergeben (Abbildung 33-11).	
Verwandte Use Cases	Use Case #05, Use Case #07, Use Case #08, Use Case #09
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Gültige, nicht korrupte SWFC RWU	Die SWF Repräsentation wird generiert und der Message Container wird initialisiert und beide werden an den SWF Interpreter übergeben.

Tabelle 15: Use Case #06: Gültige SWFC RWU, zweiter Start des Subworkflows

13.6.3.3 Szenario: Korrupte SWFC RWU – ungültige SWF ID und / oder Versionsnummer

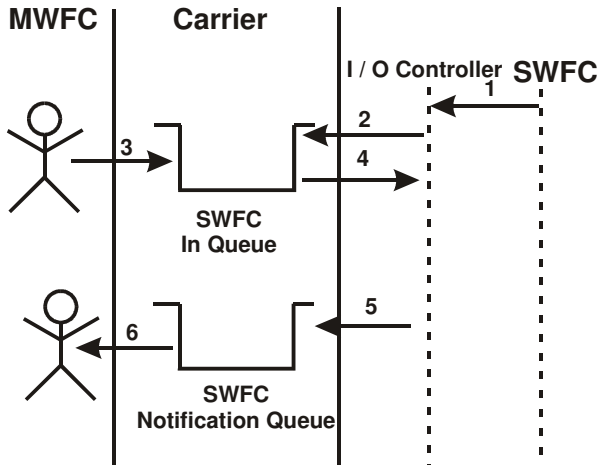

Abbildung 34: Use Case #07: Korrupte SWFC RWU – ungültige SWF ID und / oder Versionsnummer

Use Case #07	**Initialisierung eines Subworkflows**
Szenario Name	Korrupte SWFC RWU – ungültige SWF ID und / oder Versionsnummer
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die erforderlichen Initialisierungen, die für die Ausführung eines Subworkflows benötigt werden.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWFC ist bereit und verfügbar.
Eingabedaten von MFC	SWFC RWU: • 999 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • empty (cardBalance) • empty (trackingItemList)
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ERROR: INVALID SWF_ID OR INVALID VERSION_NO)
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
Lokale Daten vor Szenario	Lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	Lokale Daten wurden noch nicht initialisiert
Use Case Beschreibung	

Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs) (Abbildung 34-1 und Abbildung 34-2) als Datenpuffer. Die Struktur dieser SWFC RWU ist (wie in Abbildung 11 dargestellt) folgendermaßen: SWF ID, gefolgt von SWF Versionsnummer, gefolgt von globalen Daten. Wenn der MFC eine SWFC RWU in die In Queue des SWFC legt wird diese über den I / O Controller eingelesen, der erkennt, dass die SWF ID und / oder die SWF Versionsnummer ungültig ist und über das Notification API eine Fehlermeldung an den MFC zurückgibt (Abbildung 34-3 bis Abbildung 34-6). Anschließend wartet der I / O Controller wieder erneut auf ankommende Aufträge.	
Verwandte Use Cases	Use Case #05, Use Case #06, Use Case #08, Use Case #09
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Korrupte SWFC RWU – ungültige SWF ID oder Versionsnummer	Der SWFC sendet über das Notification API eine Fehlermeldung an den MFC

Tabelle 16: Use Case #07: Korrupte SWFC RWU – ungültige SWF ID und / oder Versionsnummer

13.6.3.4 Szenario: Korrupte RWU – ungültige Messagedaten

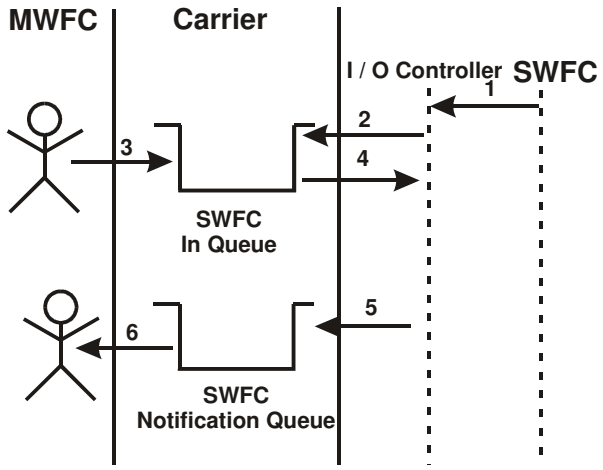

Abbildung 35: Use Case #08: Korrupte RWU – ungültige Messagedaten

Use Case #08	**Initialisierung eines Subworkflows**
Szenario Name	Korrupte SWFC RWU – ungültige Messagedaten
Benutzter SWF	Abbildung 24

Use Case Übersicht	Dieser Use Case beschreibt die erforderlichen Initialisierungen, die für die Ausführung eines Subworkflows benötigt werden.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWFC ist bereit und verfügbar.
Eingabedaten von MFC	SWFC RWU: • 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • empty (cardBalance)
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ERROR: INVALID SWFC RWU OR MISSING DATA IN SWFC RWU)
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
Lokale Daten vor Szenario	Lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	Lokale Daten wurden noch nicht initialisiert
Use Case Beschreibung	
Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs) (Abbildung 35-1 und Abbildung 35-2) als Datenpuffer. Die Struktur dieser SWFC RWU ist (wie in Abbildung 11 dargestellt) folgendermaßen: SWF ID, gefolgt von SWF Versionsnummer, gefolgt von globalen Daten. Wenn der MFC eine SWFC RWU in die In Queue des SWFC legt, wird diese über den I / O Controller eingelesen, der erkennt, dass die Messagedaten ein ungültiges Layout besitzen (Kreditkartennummer und Tracking Datenfeld fehlen) und über das Notification API eine Fehlermeldung an den MFC zurückgibt (Abbildung 35-3 bis Abbildung 35-6). Anschließend wartet der I / O Controller wieder wie vorher auf neu ankommende Aufträge.	

Verwandte Use Cases	Use Case #05, Use Case #06, Use Case #07, Use Case #09
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Korrupte SWFC RWU – ungültige Messagedaten	Der SWFC sendet über das Notification API eine Fehlermeldung an den MFC

Tabelle 17: Use Case #08: Korrupte RWU – ungültige Messagedaten

13.6.3.5 Szenario: Nicht auffindbare globale und / oder lokale Daten

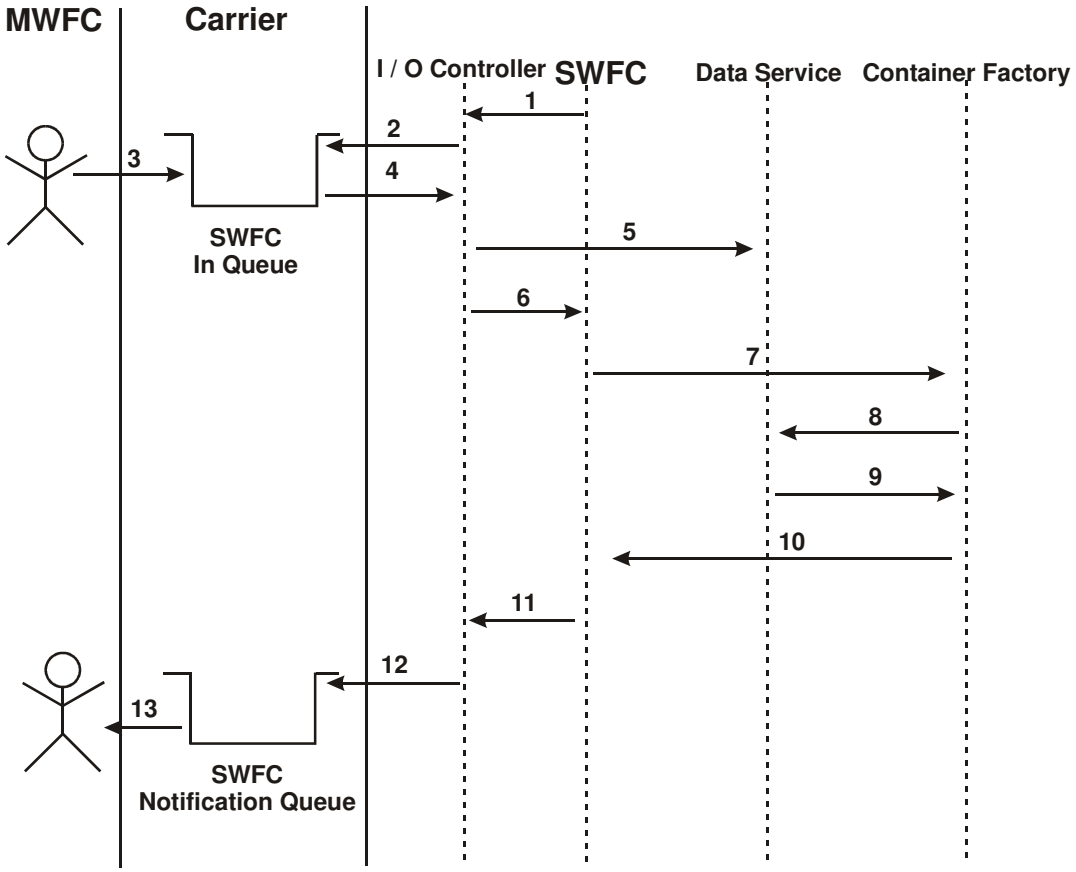

Abbildung 36: Use Case #09: Nicht auffindbare globale und / oder lokale Daten

Use Case #09	**Initialisierung eines Subworkflows**
Szenario Name	Nicht auffindbare globale und / oder lokale Daten
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die erforderlichen Initiali-

	sierungen, die für die Ausführung eines Subworkflows benötigt werden.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWFC
Vorbedingungen	• Der SWFC ist bereit und verfügbar. • Die Data Description und die Data Section Copystruktur ist bereits vom I / O Controller zwischengespeichert. • Die SWF Repräsentation ist bereits vom SWFC zwischengespeichert.
Eingabedaten von MFC	SWFC RWU: • 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 9999999999 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • empty (cardBalance) • empty (trackingItemList)
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ERROR: CANNOT INITIALIZE MESSAGE CONTAINER)
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
Lokale Daten vor Szenario	lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	lokale Daten wurden noch nicht initialisiert
Use Case Beschreibung	

Der SWFC wartet über den I / O Controller an der SWFC Input Queue auf neu ankommende Aufträge (SWFC RWUs) (Abbildung 36-1 und Abbildung 36-2). Wenn der MFC eine SWFC RWU in die In Queue des SWFC legt, liest der I / O Controller diese von der Queue (Abbildung 36-3 bis Abbildung 36-4). Anschließend gibt der I / O Controller die entsprechende zwischengespeicherte Data Description weiter an den Data Service, der diese für den Zeitraum der Ausführung dieses Subworkflows zwischenspeichert (Abbildung 36-5). Daraufhin erzeugt der I / O Controller aus der SWFC RWU mit Hilfe der entsprechenden zwischengespeicherten Data Section Copystruktur ein Java RWU Objekt und übergibt dieses an den SWFC (Abbildung 36-6). Dieser gibt das Java RWU Objekt an die Container Factory, die damit den Message Container initialisiert, der danach versucht fehlende Felder mit Hilfe des Data Service zu füllen, was zu einer Fehlermeldung führt, da in der Datenbank keine Kreditkartendaten für eine Kreditkarte mit der Nummer 9999999999 existiert (Abbildung 36-7 bis Abbildung 36-9). Daraufhin sendet die Container Factory eine Fehlermeldung an den SWFC, der über das Notification API des I / O Controller eine Fehlermeldung an den MFC zurückgibt (Abbildung 36-10 bis Abbildung 36-13).	
Verwandte Use Cases	Use Case #05, Use Case #06, Use Case #07, Use Case #08
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Nicht auffindbare globale und / oder lokale Daten	Der SWFC sendet über das Notification API eine Fehlermeldung an den MFC

Tabelle 18: Use Case #09: Korrupte SWFC RWU – ungültige Messagedaten

13.6.4 Use Case: Verwaltung der globalen und lokalen Daten

13.6.4.1 Szenario: Ausführung einer Java SWF Aktivität

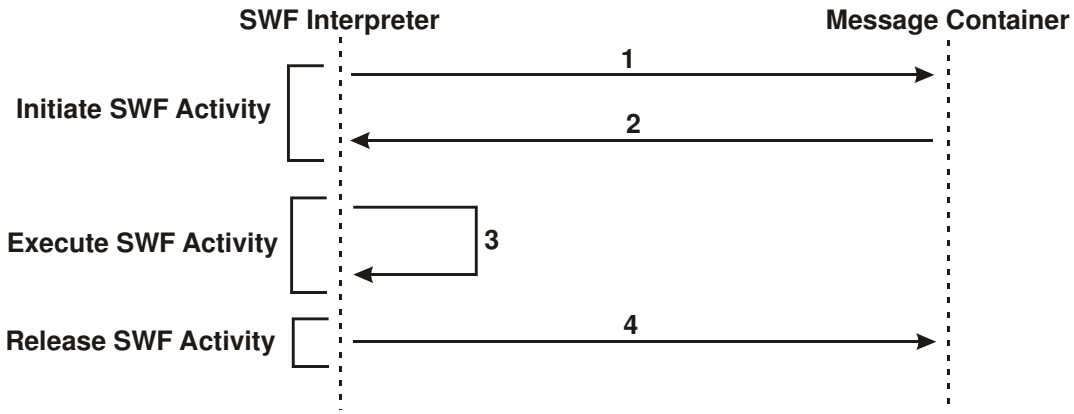

Abbildung 37: Use Case #10: Ausführung einer SWF Aktivität

Use Case #10	**Message Handling**
Szenario Name	Ausführung einer Java SWF Aktivität
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die Aspekte, die von der Verwaltung der globalen und lokalen Daten betroffen sind.
Zustand Beginn	Ready SWF
Zustand Ende	Ready SWF
Vorbedingungen	Die nächste Aktivität, die der SWF Interpreter ausführt ist checkCreditCard.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	• empty (cardStatus)

	• 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
In der Initiate SWF Activity-Phase wird mit Hilfe der Message Signatur eine lokale Sicht des Message Containers erstellt (Abbildung 37-1 und Abbildung 37-2). In der Execute SWF Activity-Phase wird die checkCreditCard-Aktivität vom SWF Interpreter mit der lokalen Sicht als Parameter aufgerufen (Abbildung 37-3) und endet mit dem Return Code RC_OK. Dann werden in der Release SWF Activity-Phase die geänderten lokalen Daten (cardStatus) in der lokalen Sicht wieder mit den Daten im Message Container synchronisiert (Abbildung 37-4). Da für die checkCreditCard-Aktivität keine Plug-Ins registriert sind, wird die Ausführung des Subworkflows anschließend normal fortgesetzt.	
Verwandte Use Cases	Use Case #02, Use Case #11, Use Case #12, Use Case #13
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Die Kreditkarte ist o.k.	CardStatus = STATUS_OK
Die Nummer der Kreditkarte ist ungültig	CardStatus = STATUS_INVALID_NUMBER
Der Name der Kreditkarte ist ungültig	CardStatus = STATUS_INVALID_NAME
Die Gültigkeitsdauer der Kreditkarte ist überschritten	CardStatus = STATUS_EXPIRED
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 19: Use Case #10: Ausführung einer SWF Aktivität

13.6.4.2 Szenario: Rückgabe des Message Containers an den MFC

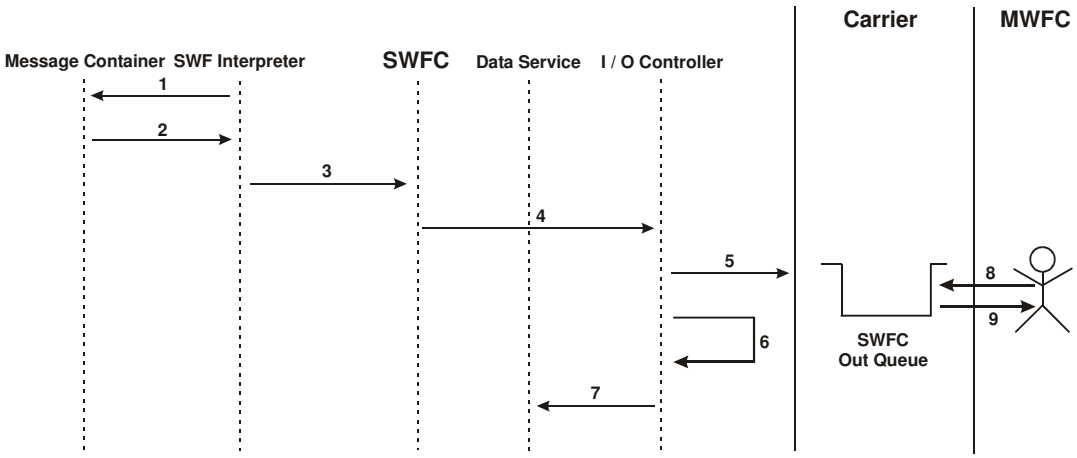

Abbildung 38: Use Case #11: Rückgabe des Message Containers an den MFC

Use Case #11	**Message Handling**
Szenario Name	Rückgabe des Message Containers an den MFC
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die Aspekte, die von der Verwaltung der globalen und lokalen Daten betroffen sind.
Zustand Beginn	Finished SWF
Zustand Ende	Ready SWFC
Vorbedingungen	Der SWF Interpreter hat gerade die successNotificationActivity-Aktivität erfolgreich ausgeführt und in den Zustand Finished SWF gewechselt.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	SWFC RWU: • 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo)

	• 250.00 (cardBalance) • 1:“credit” (trackingItemList)
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:“credit” (trackingItemList)
SWFC RWU nach Szenario	SWFC RWU des nachfolgenden Subworkflows wurde noch nicht über den I / O Controller angenommen und in ein Java RWU Objekt umgewandelt.
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 150.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	Keine
Use Case Beschreibung	
Der Message Container kann die globalen Daten als Java RWU Objekt ausgeben. So wird am Ende des Subworkflows aus dem Message Container ein Java RWU Objekt erzeugt (Abbildung 38-1 und Abbildung 38-2), welches dann wieder an den SWFC als Resultat des fehlerfreien Subworkflows zurückgegeben wird (Abbildung 38-3). Dieser gibt das Java RWU Objekt an den I / O Controller weiter, der dieses wieder in das entsprechende Datenpufferformat für die Out Queue überführt und das Resultat (die Ergebnis SWFC RWU) für den MFC in die Out Queue legt (Abbildung 38-4 bis Abbildung 38-5). Anschließend löscht der I / O Controller die zu Beginn gespeicherte SWFC RWU, da sie nun bearbeitet wurde und signalisiert anschließend dem Data Service, das der letzte Savepoint gelöscht werden soll (Abbildung 38-6 und Abbildung 38-7), damit im Fall eines Systemcrash der SWF nicht erneut ausgeführt wird. Sobald die Nachricht in der Notification Queue ist, kann der MFC sofort unabhängig vom weiteren Verlauf des Subworkflows darauf zugreifen (Abbildung 38-8 bis Abbildung 38-9).	
Verwandte Use Cases	Use Case #10, Use Case #12, Use Case #13
Bedingungen, die das Resultat beeinflussen	Resultat(e)

Keine	Keine

Tabelle 20: Use Case #11: Rückgabe der SWFC RWU an den MFC

13.6.4.3 Szenario: Persistierung der globalen und lokalen Daten durch Rückschreiben in 1..n Datenbanken bei commitTx

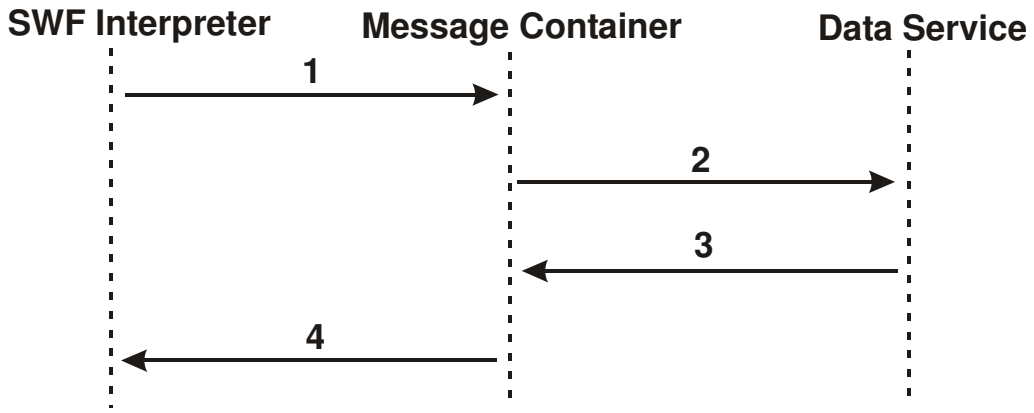

Abbildung 39: Use Case #12: Persistierung der globalen und lokalen Daten durch Rückschreiben in 1..n Datenbanken bei commitTx

Use Case #12	**Message Handling**
Szenario Name	Persistierung der globalen und lokalen Daten durch Rückschreiben in 1..n Datenbanken bei commitTx
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die Aspekte, die von der Verwaltung der globalen und lokalen Daten betroffen sind.
Zustand Beginn	Ready SWF
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter hat gerade die debit-Aktivität erfolgreich ausgeführt.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number)

	• 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 150.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 150.00 (balanceDestinationAccount)
Use Case Beschreibung	
Der SWF Interpreter signalisiert dem Message Container, dass ein commitTx erfolgen soll und übergibt ihm zusätzliche Statusinformationen (Abbildung 39-1), woraufhin der Message Container mit Hilfe des Data Service die geänderten globalen und lokalen Daten (cardBalance und balanceDestinationAccount) in die 1..n Datenbanken rückschreibt und die globalen und lokalen Daten und die Statusinformationen über einen Savepoint absichert (Abbildung 39-2). Der Data Service signalisiert dem Message Container anschließend den Erfolg der Operation, der wiederum dem SWF Interpreter die erfolgreiche Durchführung des commitTx meldet (Abbildung 39-3 und Abbildung 39-4).	
Verwandte Use Cases	Use Case #10, Use Case #11, Use Case #13
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Keine	Keine

Tabelle 21: Use Case #12: Persistierung der globalen und lokalen Daten durch Rückschreiben in 1..n Datenbanken bei commitTx

13.6.4.4 Szenario: Persistierung der globalen und lokalen Daten durch Schreiben eines Savepoints

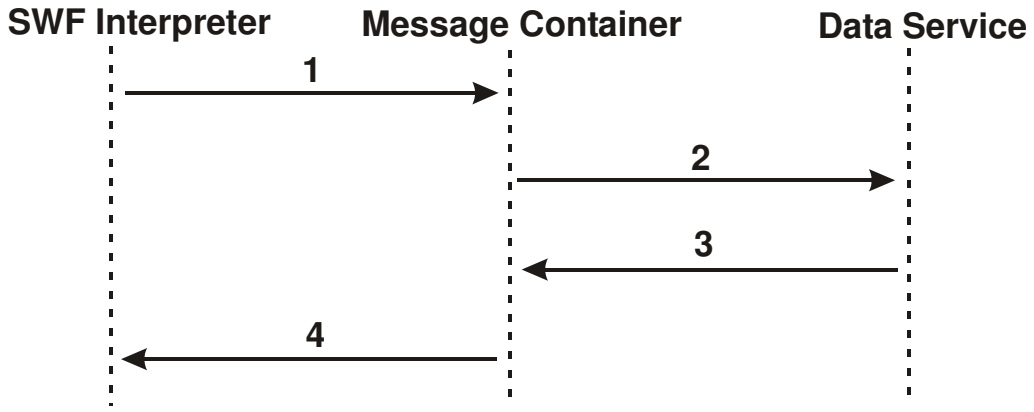

Abbildung 40: Use Case #13: Persistierung der globalen und lokalen Daten durch Schreiben eines Savepoints

Use Case #13	**Message Handling**
Szenario Name	Persistierung der globalen und lokalen Daten durch Schreiben eines Savepoints.
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die Aspekte, die von der Verwaltung der globalen und lokalen Daten betroffen sind.
Zustand Beginn	Ready SWF
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter hat gerade die checkCreditCard-Aktivität erfolgreich ausgeführt.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number)

	• 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Der SWF Interpreter signalisiert dem Message Container, dass ein Savepoint geschrieben werden soll und übergibt ihm zusätzliche Statusinformationen (Abbildung 40-1). Der Message Container speichert die Daten (global, lokal und Statusinformationen) lokal zwischen und persistiert zusätzlich das Java RWU Objekt (welches der Message Container liefert) mit den aktuellen Daten und die Statusinformationen mit Hilfe des Data Service (hier allerdings immer nur den aktuellen Savepoint!), um im Fall eines Systemcrashs wieder auf den letzten Savepoint zurücksetzen zu können (Abbildung 40-2). Der Data Service signalisiert dem Message Container anschließend den Erfolg der Operation, der wiederum dem SWF Interpreter die erfolgreiche Durchführung des Schreibens des Savepoints meldet (Abbildung 40-3 und Abbildung 40-4).	
Verwandte Use Cases	Use Case #10, Use Case #11, Use Case #12
Bedingungen, die das Resultat beeinflussen	Resultat(e)

Keine	Keine

Tabelle 22: Use Case #13: Message Handling-Persistierung der globalen und lokalen Daten durch Schreiben eines Savepoints

13.6.5 Use Case: Aufruf von COBOL Aktivitäten

13.6.5.1 Ausführung einer COBOL SWF Aktivität

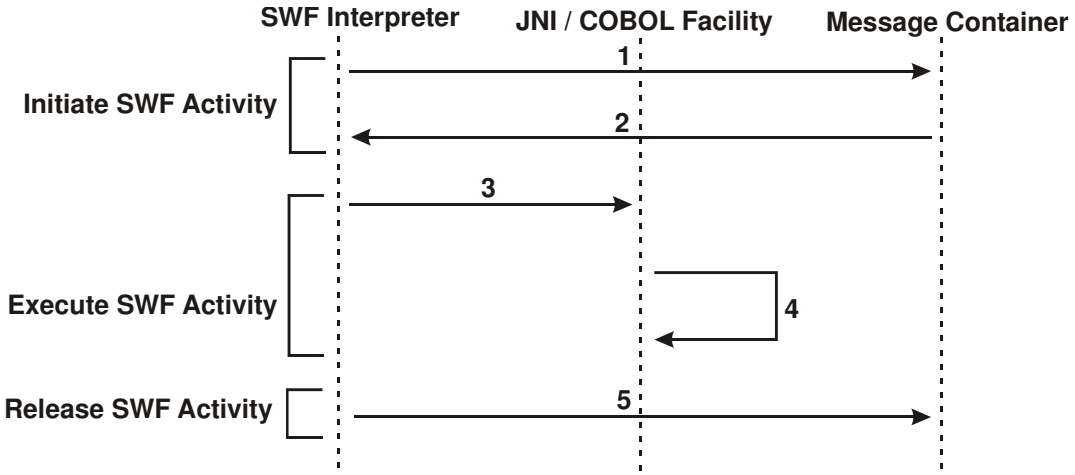

Abbildung 41: Use Case #14: Ausführung einer COBOL Aktivität

Use Case #14	**Aufruf von COBOL Aktivitäten**
Szenario Name	Ausführung einer COBOL Aktivität
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt den Aufruf von COBOL Aktivitäten.
Zustand Beginn	Ready SWF
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter hat gerade die credit-Aktivität erfolgreich ausgeführt.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number)

	• 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 150.00 (balanceDestinationAccount)
Use Case Beschreibung	

In der Initiate SWF Activity-Phase wird über das COBOL API des Message Containers ein Java COBOL Objekt erzeugt (Abbildung 41-1 und Abbildung 41-2), welches ein Feld für balanceDestinationAccount und ein zusätzliches Feld für den Return Code enthält. Dieses Java COBOL Objekt wird (per Referenz) in der Execute SWF Activity-Phase zusammen mit dem Namen der entsprechenden C Bridge (SWF23.dll) und dem Namen der COBOL-Aktivität (debit) an die COBOL / JNI Facility als Parameter übergeben (Abbildung 41-3), die diese C Bridge aufruft. In der C Bridge wird dann der Datenpuffer für die debit-Aktivität über das Java COBOL Objekt als C Struct bestehend aus einem double Feld mit dem Kontostand des Empfängers (balanceDestinationAccount) und einem int Feld für den Return Code (0 für RC_OK) initialisiert. Dieser gefüllte Datenpuffer (C Struct) wird dann an die COBOL Aktivität als Parameter übergeben, welche die Daten entsprechend ändert (das balanceDestinationAccount-Feld um 100 erhöht und bei erfolgreicher Ausführung den Return Code unverändert bei 0 für RC_OK belässt). Nach der Ausführung der COBOL-Aktivität wird in der C Bridge das Java COBOL Objekt mit den u.U. geänderten Daten im Datenpuffer synchronisiert (Abbildung 41-4). Da das Java COBOL Objekt per Referenz übergeben wurde, sind eventuelle Änderungen automatisch auch außen (im SWF Interpreter) sichtbar. Anschließend wird dann in der Release SWF Activity-Phase das u.U. geänderte Java COBOL Objekt mit den globalen und lokalen Daten im Message Container synchronisiert (Abbildung 41-5). Da für diese Aktivität keine Plug-Ins registriert sind wird die Ausführung des Subworkflows anschließend normal fortgesetzt.	
Verwandte Use Cases	Keine
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 23: Use Case #14: Ausführung einer COBOL Aktivität

13.6.6 Use Case: Plug-In Aktivitäten

13.6.6.1 Ausführung von Plug-In Aktivitäten

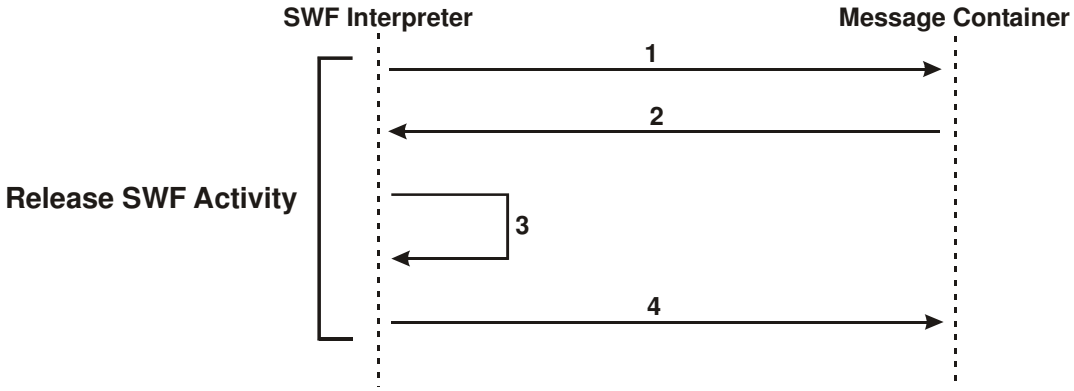

Abbildung 42: Use Case #15: Ausführung von Plug-In Aktivitäten

Use Case #15	**Plug-In Aktivitäten**
Szenario Name	Ausführung von Plug-In Aktivitäten
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt die Ausführung von Plug - In Aktivitäten
Zustand Beginn	Released SWF Activity
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter hat gerade die credit-Aktivität erfolgreich ausgeführt (Return Code RC_OK).
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance)

	• empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Anhand der Message Signatur des Tracking Plug-Ins wird eine lokale Sicht des Message Containers erstellt (Abbildung 42-1 und Abbildung 42-2). Anschließend wird sie vom SWF Interpreter mit der lokalen Sicht und dem Namen der gerade ausgeführten Aktivität (credit) als Parameter aufgerufen (Abbildung 42-3) und endet mit dem Return Code RC_OK. Dann werden die geänderten Daten (trackingItemList) in der lokalen Sicht wieder mit den Daten im Message Container synchronisiert (Abbildung 42-4) und der SWF Interpreter setzt die Ausführung des Subworkflows mit der nächsten Aktivität (debit) fort.	
Verwandte Use Cases	Keine
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Return Code	Siehe Kapitel 6.3: Ausführung von Plug-In Aktivitäten

Tabelle 24: Use Case #15: Ausführung von Plug-In Aktivitäten

13.6.7 Use Case: Fehlerbehandlung des Subworkflows

13.6.7.1 Szenario: Recover

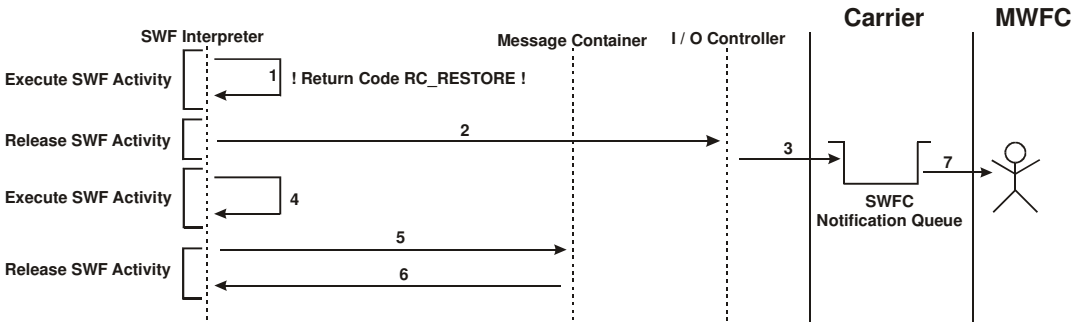

Abbildung 43: Use Case #16: Recover

Use Case #16	**Fehlerbehandlung des Subworkflows**
Szenario Name	Recover
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt alle Aspekte, die mit dem transaktionalen Kontext des SWF bzw. mit der Fehlerbehandlung des SWF verbunden sind.
Zustand Beginn	Executing SWF Activity
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter führt gerade die checkCreditCard-Aktivität aus.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ACTIVTY CHECK_CREDIT_CARD RETURNED RC_RECOVER)
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo)

	• 150.00 (cardBalance) • empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	• empty (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Die Ausführung der checkCreditCard-Aktivität endet in der Execute SWF Activity-Phase mit dem Return Code RC_RECOVER (Abbildung 43-1). Daraufhin wird der MFC über das Notification API davon informiert, dass die Aktivität erneut gestartet wird (Abbildung 43-2 bis Abbildung 43-3) und die Aktivität wird erneut mit denselben Eingabedaten im Ausführungskontext (Execution Context) RECOVER gestartet (Abbildung 43-4). Diesmal verläuft die Ausführung der Aktivität erfolgreich (Abbildung 43-5 bis Abbildung 43-6). Sobald die Nachricht in der Notification Queue ist, kann der MFC sofort unabhängig vom weiteren Verlauf des Subworkflows darauf zugreifen (Abbildung 43-7).	
Verwandte Use Cases	Use Case #17, Use Case #18
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Die Kreditkarte ist o.k.	CardStatus = STATUS_OK
Die Nummer der Kreditkarte ist ungültig	CardStatus = STATUS_INVALID_NUMBER
Der Name der Kreditkarte ist ungültig	cardStatus = STATUS_INVALID_NAME

Die Gültigkeitsdauer der Kreditkarte ist überschritten	cardStatus = STATUS_EXPIRED
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 25: Use Case #16: Recover

13.6.7.2 Szenario: Error Workflow

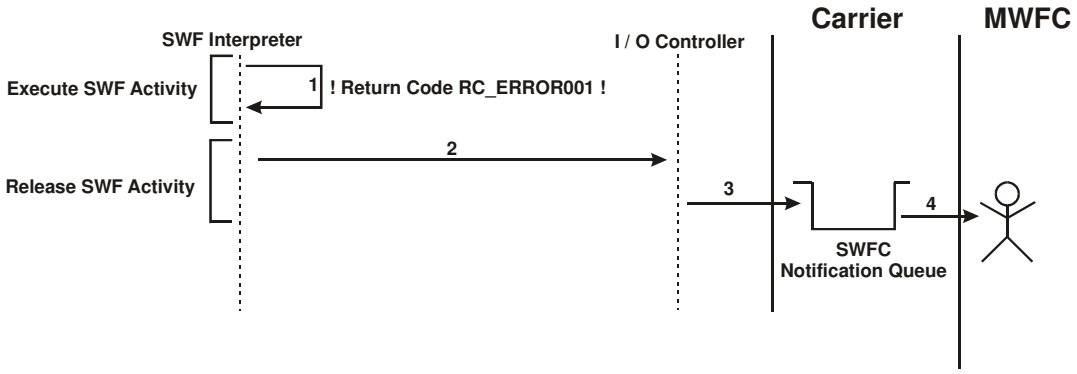

Abbildung 44: Use Case #17: Error Workflow

Use Case #17	**Fehlerbehandlung des Subworkflows**
Szenario Name	Error Workflow
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt alle Aspekte, die mit dem transaktionalen Kontext des SWF bzw. mit der Fehlerbehandlung des SWF verbunden sind.
Zustand Beginn	Executing SWF Activity
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter führt gerade die credit-Aktivität aus.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ACTIVTY CREDIT RETURNED RC_ERROR001)
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer)

	• 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Die Ausführung der credit-Aktivität endet in der Execute SWF Activity-Phase mit dem Return Code RC_ERROR001 (Abbildung 44-1). Daraufhin wird der MFC über das Notification API davon informiert, dass der Fehler ERROR001 aufgetreten ist (Abbildung 44-2 bis Abbildung 44-4) und die Ausführung des SWF wird anschließend mit dem Error Workflow fortgesetzt.	
Verwandte Use Cases	Use Case #16, Use Case #18
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 26: Use Case #17: Error Workflow

13.6.7.3 Szenario: Wiederaufnahme eines Subworkflows mit Hilfe eines Savepoints nach einem Systemcrash

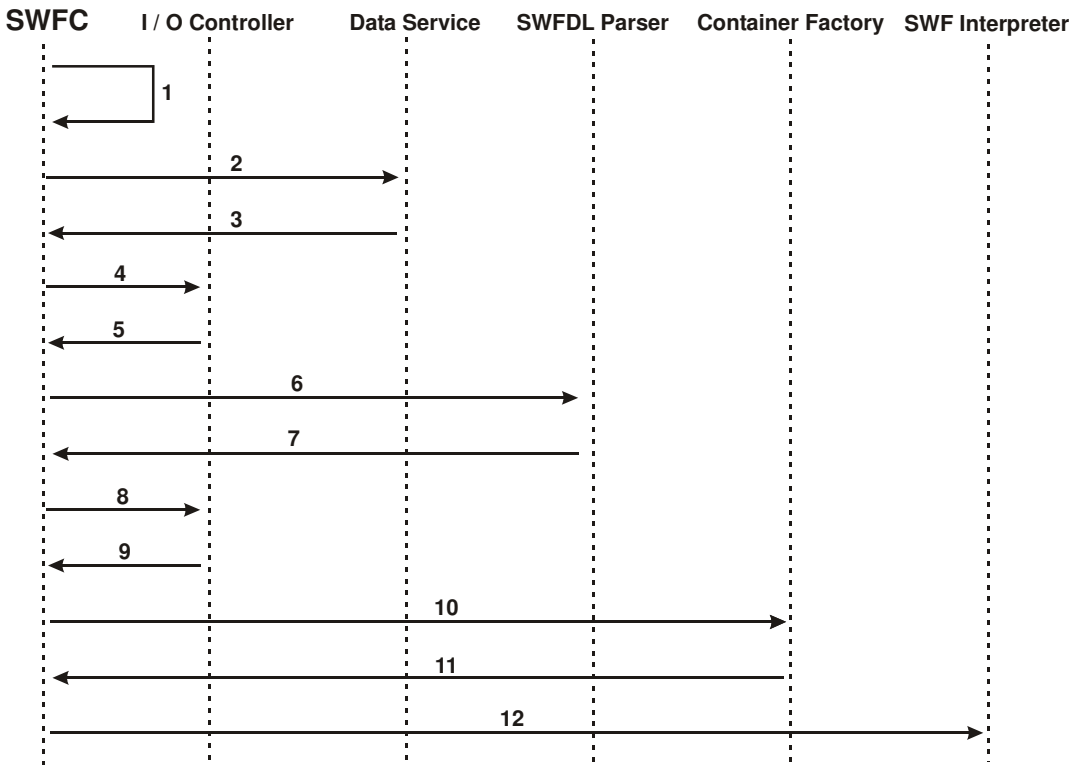

Abbildung 45: Use Case #18: Wiederaufnahme eines Subworkflows mit Hilfe eines Savepoints nach einem Systemcrash

Use Case #18	**Fehlerbehandlung des Subworkflows**
Szenario Name	Wiederaufnahme eines Subworkflows mit Hilfe eines Savepoints nach einem Systemcrash
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt alle Aspekte, die mit dem transaktionalen Kontext des SWF bzw. mit der Fehlerbehandlung des SWF verbunden sind.
Zustand Beginn	Ready SWFC
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter hat gerade die debit-Aktivität ausgeführt, als es einen Systemcrash gab. Beim Neustart des SWFC wurde geprüft, ob noch ein alter Savepoint existiert. Da dies der Fall war, wird zuerst der letzte abgebrochene SWF beendet und anschließend wird

	normal mit der Ausführung von Subworkflows fortgefahren.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Keine
SWFC RWU vor Szenario	SWFC RWU wurde noch nicht in ein Java RWU Objekt umgewandelt
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • 1:”credit” (trackingItemList)
Lokale Daten vor Szenario	Lokale Daten wurden noch nicht initialisiert
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	

Der SWFC prüft beim Start ob noch ein alter Savepoint existiert. Dies ist der Fall und so prüft er mit Hilfe des Data Service beim Start, ob noch ein alter Savepoint existiert (Abbildung 45-1). Dies ist der Fall und so liest er diesen mit Hilfe des Data Service von der Festplatte bzw. der SWFC Datenbank im SWFC RWU Format (Abbildung 45-2 bis Abbildung 45-3). Mit Hilfe des I / O Controllers wird dieser dann anschließend in einen Savepoint konvertiert (Abbildung 45-4 bis Abbildung 45-5). Der SWFC gibt dann die SWF ID und SWF Versionsnummer weiter an den SWFDL Parser, der die entsprechende SWFDL-Beschreibung einliest, parst und daraus eine SWF Repräsentation erstellt, die er an den SWFC zurückgibt, der diese zwischenspeichert (Abbildung 45-6 bis Abbildung 45-7). Anschließend konvertiert der SWFC mit Hilfe des I / O Controllers den Savepoint in eine Java RWU, die er an die Container Factory weitergibt, die daraus den Message Container initialisiert und an den SWFC zurückgibt (Abbildung 45-8 bis Abbildung 45-11). Zum Schluss wird der Message Container, die entsprechenden Statusinformationen, Finite State Machine Zustand, Ausführungskontext (Execution Context) und die SWF Repräsentation an den SWF Interpreter übergeben (Abbildung 45-12), welcher die Ausführung des Subworkflows anhand der Statusinformationen an der richtigen Position mit dem richtigen Ausführungskontext im richtigen FSM Zustand fortsetzt. Die nächste Aktivität, die ausgeführt wird, ist somit die credit-Aktivität.	
Verwandte Use Cases	Use Case #03, Use Case #16, Use Case #17
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Keine	Keine

Tabelle 27: Use Case #18: Wiederaufnahme eines Subworkflows mit Hilfe eines Savepoints nach einem Systemcrash

13.6.8 Use Case: Transaktionale Steuerung des Subworkflows

13.6.8.1 Szenario: RollbackTx

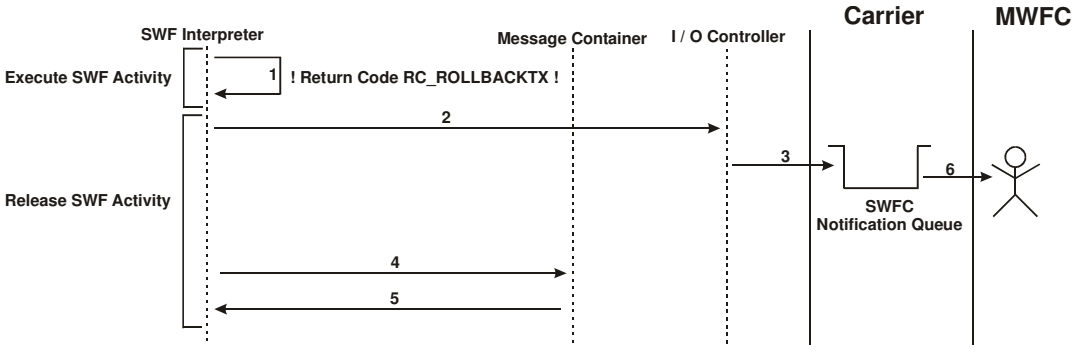

Abbildung 46: Use Case #19: RollbackTx

Use Case #19	**Transaktionale Steuerung des Subworkflows**
Szenario Name	RollbackTx
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt alle Aspekte, die mit dem transaktionalen Kontext des SWF bzw. mit der Fehlerbehandlung des SWF verbunden sind.
Zustand Beginn	Executing SWF
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter führt gerade die debit-Aktivität aus.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ACTIVTY DEBIT RETURNED RC_ROLLBACKTX)
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 250.00 (cardBalance) • 1:”credit” (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance)

	• 1:”credit” (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• empty (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Die Ausführung der debit-Aktivität endet in der Execute SWF Activity-Phase mit dem Return Code RC_ROLLBACKTX (Abbildung 46-1). Daraufhin wird der MFC über das Notification API davon informiert, dass ein rollbackTx stattfindet (Abbildung 46-2 bis Abbildung 46-3) und der SWF Interpreter signalisiert dem Message Container, dass die globalen und lokalen Daten auf den Beginn des SWF zurückgesetzt werden müssen (Abbildung 46-4), da dies der letzte beginTx bzw. commitTx-Punkt ist. Der Message Container setzt die globalen und lokalen Daten mit Hilfe der lokal zwischengespeicherten Daten beim letzten commitTx auf die entsprechenden Werte zurück und liefert als Resultat die (ebenfalls zwischengespeicherten) Statusinformationen für den SWF Interpreter, damit dieser die Ausführung des Subworkflows wieder von Anfang an beginnen kann (Abbildung 46-5). Die nächste Aktivität, die im Ausführungskontext (Execution Context) ROLLBACKTX ausgeführt wird, ist somit checkCreditCard. Sobald die Nachricht in der Notification Queue ist, kann der MFC sofort unabhängig vom weiteren Verlauf des Subworkflows darauf zugreifen (Abbildung 46-6).	
Verwandte Use Cases	Use Case #20
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 28: Use Case #19: RollbackTx

13.6.8.2 Szenario: RestoreTx

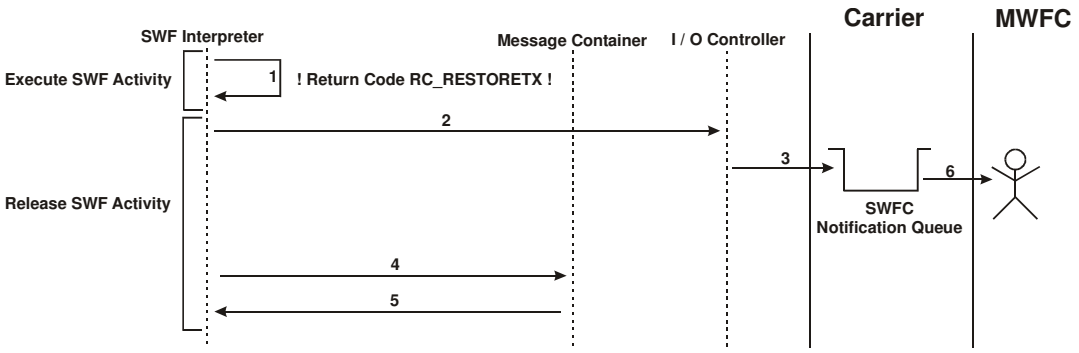

Abbildung 47: Use Case #20: RestoreTx

Use Case #20	**Transaktionale Steuerung des Subworkflows**
Szenario Name	RestoreTx
Benutzter SWF	Abbildung 24
Use Case Übersicht	Dieser Use Case beschreibt alle Aspekte, die mit dem transaktionalen Kontext des SWF bzw. mit der Fehlerbehandlung des SWF verbunden sind.
Zustand Beginn	Executing SWF Activity
Zustand Ende	Ready SWF
Vorbedingungen	Der SWF Interpreter führt gerade die credit-Aktivität aus.
Eingabedaten von MFC	Keine
Rückgabedaten an MFC	Notification Message an den MFC mit Fehlerbeschreibung (z.B. ACTIVTY CREDIT RETURNED RC_RESTORE)
SWFC RWU vor Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance)

	• empty (trackingItemList)
SWFC RWU nach Szenario	• 23 (SWF ID) • 1.00 (SWF Versionsnummer) • 1234567890 (creditCard.number) • 07/01 (creditCard.expirationDate) • Max Mustermann (creditCard.name) • 100.00 (amount) • 1234567 (destinationAccountNo) • 150.00 (cardBalance) • empty (trackingItemList)
Lokale Daten vor Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Lokale Daten nach Szenario	• STATUS_OK (cardStatus) • 50.00 (balanceDestinationAccount)
Use Case Beschreibung	
Die Ausführung der credit-Aktivität endet in der Execute SWF Activity-Phase mit dem Return Code RC_RESTORE (Abbildung 47-1). Daraufhin wird der MFC über das Notification API davon informiert, dass auf den letzten Savepoint zurückgesetzt wird (Abbildung 47-2 bis Abbildung 47-3) und der SWF Interpreter signalisiert dem Message Container, dass die globalen und lokalen Daten auf den Punkt vor dem Beginn der Aktivität zurückgesetzt werden müssen (, da dies der letzte Savepoint ist) (Abbildung 47-4). Der Message Container setzt die globalen und lokalen Daten mit Hilfe der lokal zwischengespeicherten Daten auf die entsprechenden Werte zurück und liefert als Resultat die (ebenfalls zwischengespeicherten) Statusinformationen für den SWF Interpreter, damit dieser die Ausführung des Subworkflows wieder vor dem Beginn der credit-Aktivität aufnehmen kann (Abbildung 47-5). Die nächste Aktivität, die im Ausführungskontext (Execution Context) RESTORETX ausgeführt wird, ist somit credit. Sobald die Nachricht in der Notification Queue ist, kann der MFC sofort unabhängig vom weiteren Verlauf des Subworkflows darauf zugreifen (Abbildung 47-6).	
Verwandte Use Cases	Use Case #19
Bedingungen, die das Resultat beeinflussen	Resultat(e)
Return Code	Siehe Kapitel 7: Fehlerbehandlung des Subworkflows

Tabelle 29: Use Case #20: RestoreTx

13.7 Performance-Test

13.7.1 Überblick und Konfiguration des Testsystems

Die in [BEY] beschriebene Konfiguration des Testsystems diente als Basis für die im Rahmen dieser Diplomarbeit durchgeführten Performance Messungen. Aus diesem Grund wird die dort dargestellte Konfiguration hier ebenfalls beschrieben.

Beide in Abbildung 48 dargestellte Plattformen liefen als Gäste unter dem z/VM-Betriebssystem und hatten je eine dedizierte CPU eines z900-Rechners (Modell 2064-109) zur Verfügung. Durch die Dedizierung ist sichergestellt, dass die z/VM-Gäste auf exakt die Rechenleistung einer physikalischen CPU zurückgreifen können. Während dem z/OS-Gast zwei Gigabyte (GB) Hauptspeicher zur Verfügung standen, wurde dem zLinux-Gast ein GB zugewiesen.

Des weiteren wurde ein Enterprise Storage System (ESS) vom Typ ESS 2105-F20 verwendet. Von diesem System wurden 19 Festplatten zu je 2,8 GB benutzt. Die Verbindung zum ESS wurde durch acht Enterprise System Connectivity (ESCON) Channels realisiert.

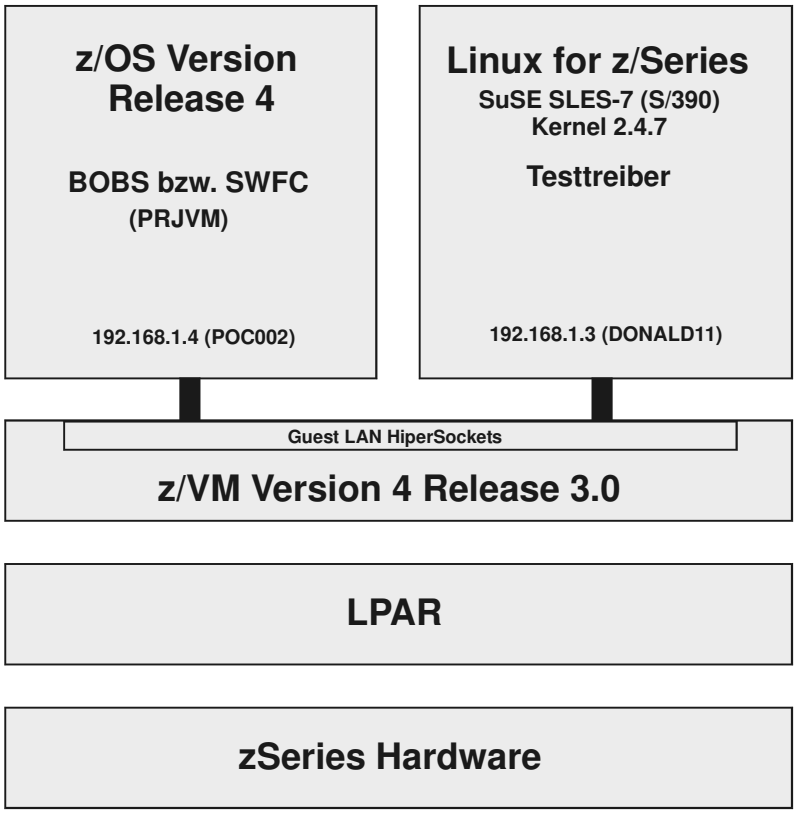

Abbildung 48: Beschreibung der Testkonfiguration

Auf der soeben beschriebenen zSeries-Hardware wurde für die Performance Messungen eine Logical Partition (LPAR) reserviert (siehe Abbildung 48). In dieser LPAR wurde ein z/VM-Betriebssystem installiert, welches drei Gäste hat: ein z/OS und zwei zLinux-Systeme.

In Abbildung 48 ist nur eines dieser Linux-Systeme dargestellt, da das andere für die im Rahmen dieser Diplomarbeit durchgeführten Messungen nicht relevant ist.

Auf dem z/OS-Gast wurde in Performance Messung Nr. 01 das in [BEY] beschriebene und für diese Messungen auf JDBC angepasste Basic Online-Banking System (BOBS) und in den Performance Messungen Nr. 02 bis 19 die Subworkflowsteuerung gestartet. In der Abbildung 48 bezeichnet die Beschriftung PRJVM die Persistent Reusable Java Virtual Machine, die als Carrier für die Subworkflowsteuerung bzw. als Ausführungsumgebung für BOBS genutzt wurde.

Der zLinux-Gast führte den zur Lastzeugung benutzten Testtreiber aus. Unter Verwendung von TPC-A Terminologie würde man ihn mit Remote Terminal Emulator bezeichnen, während das z/OS System dem System Under Test (SUT) entspricht.

Wie man in Abbildung 48 erkennen kann, kommunizierte der Lastgeber mit dem SUT über sogenannte HiperSockets. Diese wurden im Testaufbau durch die Guest LAN Technologie des z/VM-Betriebssystems realisiert.

HiperSockets ist eine mit der zSeries-Architektur eingeführte Microcode-Funktion, die unter Verwendung des Hauptspeichers eine sehr schnelle TCP/IP-Kommunikation zwischen zwei Servern ermöglicht (siehe [WHI]). Diese Option wurde verwendet, um Flaschenhälse (Bottlenecks) auszuschließen, die durch das Netzwerk verursacht werden können.

Da der Haupteinfluss auf die Performance abhängig von der CPU-Auslastung ist, wurde diese Messgröße in zwei Kategorien unterteilt:

- Die CPU Nutzung des Adressraums, inklusive Applikation (Subworkflowsteuerung bzw. BOBS),
- Die CPU Nutzung durch die Applikation im Adressraum

Zusätzlich wurde die Hauptspeicher-Auslastung gemessen.

Für jede dieser Messungen wurde anschließend der jeweilige Mittelwert und die Standardabweichung bestimmt. Um bewerten zu können, ob die Messung statistisch signifikant ist, wurden dazu hin folgende Signifikanzniveaus (prozentuale Anzahl der Messwerte, die mehr als das Doppelte der Standardabweichung vom Mittelwert abweichen) festgelegt:

- Sehr signifikant: prozentualer Anteil der abweichenden Werte ist kleiner als ein Prozent der Gesamtanzahl der Messwerte,
- Signifikant: prozentualer Anteil der abweichenden Werte ist größer als ein Prozent und kleiner als fünf Prozent der Gesamtanzahl der Messwerte,

- Marginal signifikant: prozentualer Anteil der abweichenden Werte ist größer als fünf Prozent und kleiner als zehn Prozent der Gesamtanzahl der Messwerte.

Dieser Wert wurde als Irrtumswahrscheinlichkeit bezeichnet.

13.7.2 Performance-Szenarien

13.7.2.1 Referenz Messung

13.7.2.1.1 Performance-Messung Nr. 01: TPC A

Diese Messung basiert auf einer angepassten Version (Änderung der Datenbankzugriffe von SQLJ auf JDBC) des in [BEY] beschriebenen Basic Online-Banking Systems (BOBS). Gemessen wurde die in [TPC] beschriebene TPC Benchmark™, Typ A Transaktion (siehe Kapitel 13.7.2.3: TPC A Vergleich mit Referenz Messung).

Diese Messung wurde durchgeführt, um bewerten zu können, wie viel Overhead durch den Einsatz der Subworkflowsteuerung selbst entsteht, da BOBS einen statischen Ansatz, der die transaktionalen Eigenschaften über die Datenbank absichert, bei der Implementierung des TPC A Benchmarks wählt und somit als „Performance-Obergrenze“ bei der Realisierung dieses Benchmarks unter einer PRJVM gelten kann.

Transaktionen	100.000
Transaktionen pro Sekunde	39,7

Tabelle 30: Überblick über die Performance-Messung Nr. 01

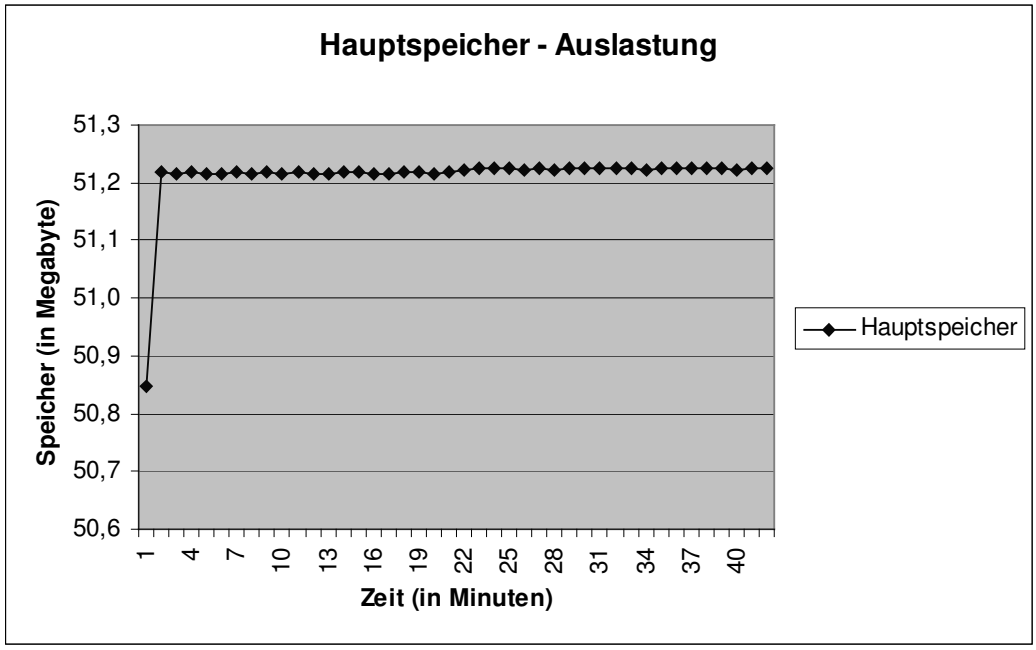

Abbildung 49: Hauptspeicher-Auslastung für Performance-Messung Nr. 01

Mittelwert	51,2
Standardabweichung	0,06
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	2,4 %
Signifikanz	Signifikant

Tabelle 31: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 01

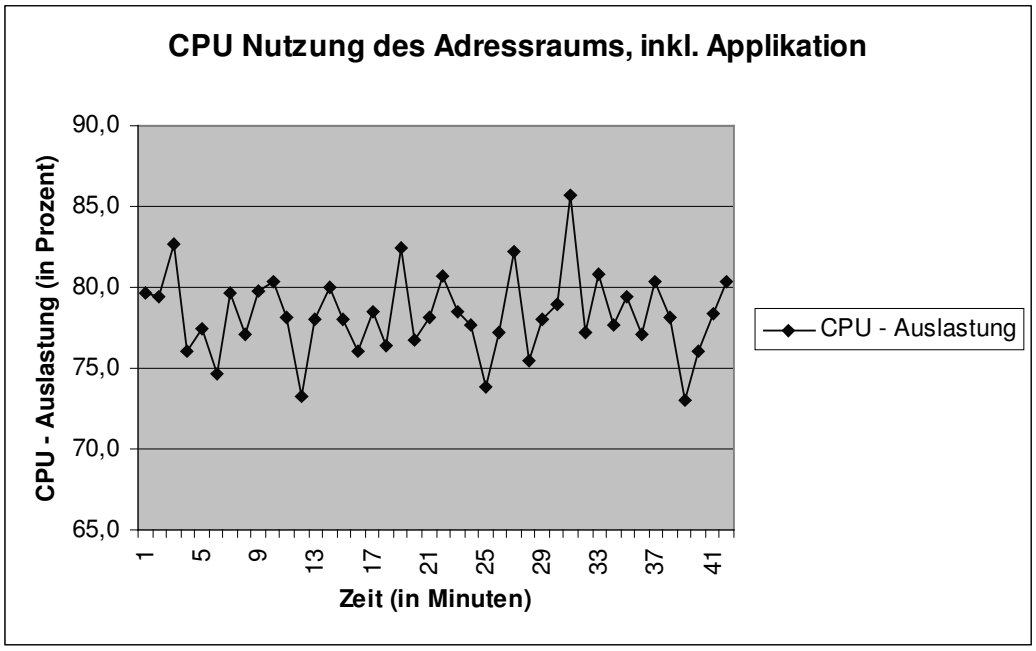

Abbildung 50: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 01

Mittelwert	78,3
Standardabweichung	2,55
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	4,8 %
Signifikanz	Signifikant

Tabelle 32: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 01

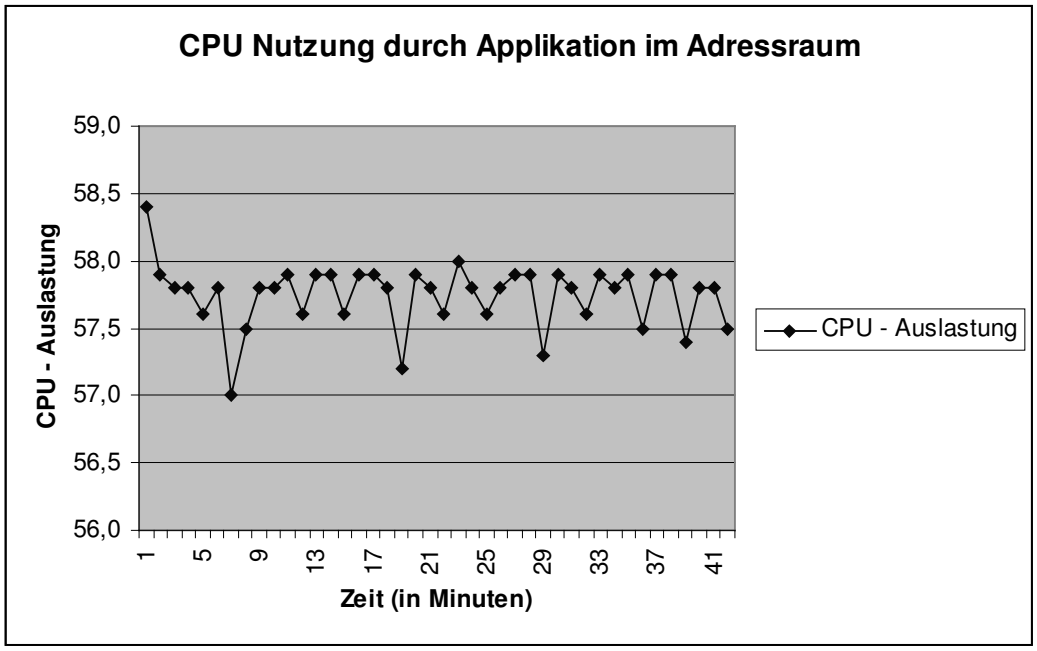

Abbildung 51: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 01

Mittelwert	57,7
Standardabweichung	0,24
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	7,1 %
Signifikanz	Marginal signifikant

Tabelle 33: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 01

13.7.2.2 Test der verschiedenen Workflow Patterns

13.7.2.2.1 Überblick

Abbildung 52: Globale Ein- und Rückgabedaten für SWF #001-SWF #005

Die folgenden Messungen fünf Messungen dienen zum Vergleich des Aufwandes für folgende Workflow Patterns (siehe Abbildung 6):

- SWF #001: Sequentiell, ohne Fehler (Java Aktivität)
- SWF #002: Sequentiell, mit Fehler (Java Aktivität)
- SWF #003: Sequentiell, mit Alternative (Java Aktivität)
- SWF #004: Iterativ sequentiell (Java Aktivität)
- SWF #005: Sequentiell ohne Fehler (COBOL Aktivität)

Hierbei gibt die Java-Aktivität A1 (SWF #001, SWF #002 und SWF #004) bzw. die COBOL Aktivität A2 (SWF #005) nur den Message String über das Notification API aus und gibt diesen dann anschließend unverändert (mit Ausnahme von SWF #002) wieder als Resultat zurück. Darüber hinaus existieren keine lokalen Daten und insbesondere werden auch keine Daten zu Beginn aus einer Datenbank initialisiert.

Da bei den Performance-Tests Nr. 02 bis 06 nur primitive Subworkflows ausgeführt werden, wird bei der Ergebnisdarstellung auf die Hauptspeicher-Auslastung verzichtet.

13.7.2.2.2Performance-Messung Nr. 02: SWF #001 – Sequentiell, ohne Fehler (Java Aktivität)

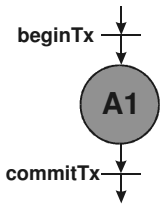

Abbildung 53: SWF #001: Sequentiell, ohne Fehler (Java Aktivität)

Die Aktivität A1 liefert immer RC_OK als Return Code an den SWF Interpreter zurück.

Transaktionen	100.000
Transaktionen pro Sekunde	97,1

Tabelle 34: Überblick über die Performance-Messung Nr. 02

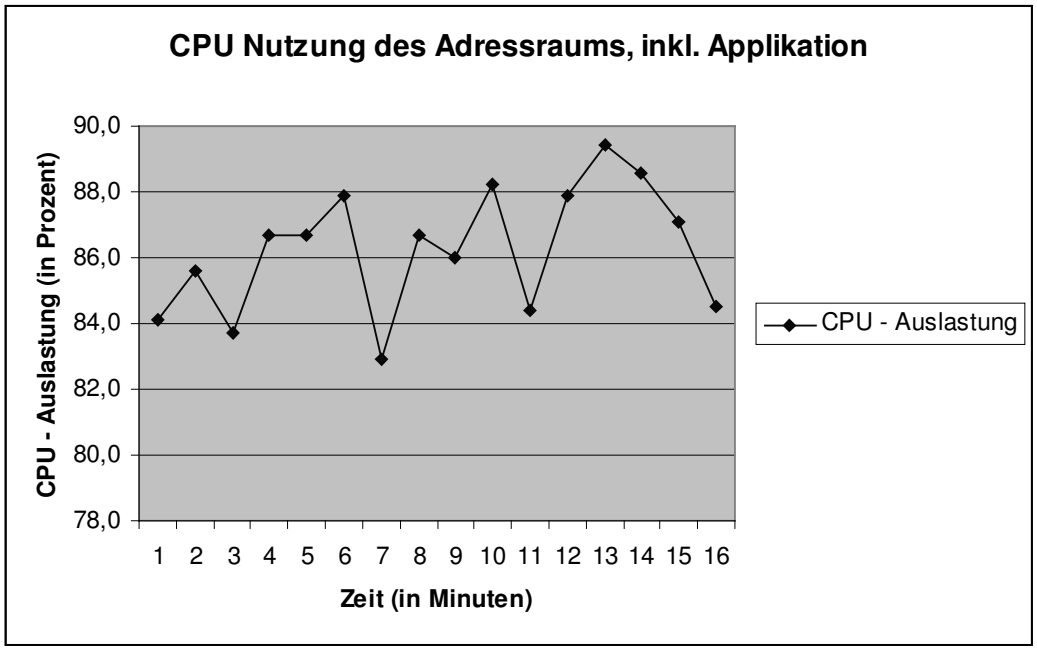

Abbildung 54: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 02

Mittelwert	86,3
Standardabweichung	1,92
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 35: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 02

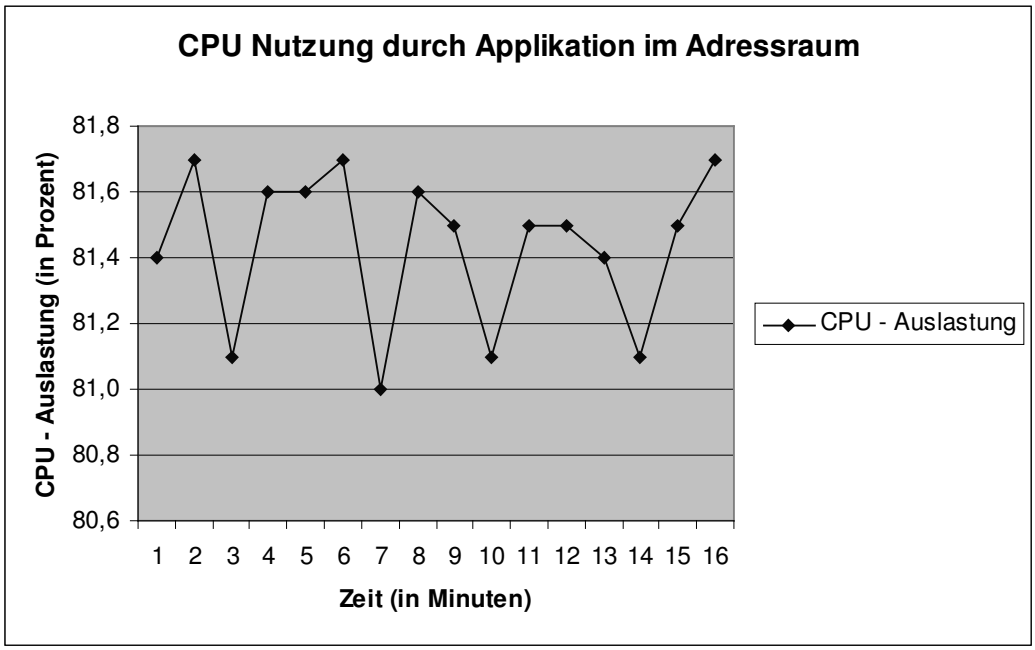

Abbildung 55: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 02

Mittelwert	81,4
Standardabweichung	0,24
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 36: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 02

13.7.2.2.3 Performance-Messung Nr. 03: SWF #002 – Sequentiell, mit Fehler (Java Aktivität)

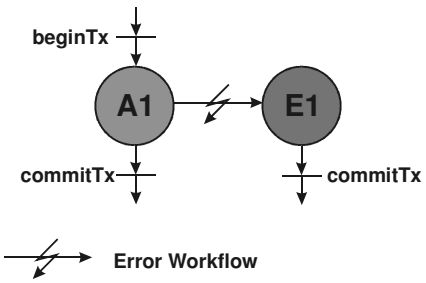

Abbildung 56: SWF #002: Sequentiell, mit Fehler (Java Aktivität)

Die Aktivität A1 liefert immer RC_ERRORxxx an den SWF Interpreter zurück, der daraufhin die Error Aktivität E1 ausführt, die über das Notification API „ERROR OCCURED“ ausgibt.

Transaktionen	100.000
Transaktionen pro Sekunde	102,3

Tabelle 37: Überblick über die Performance-Messung Nr. 03

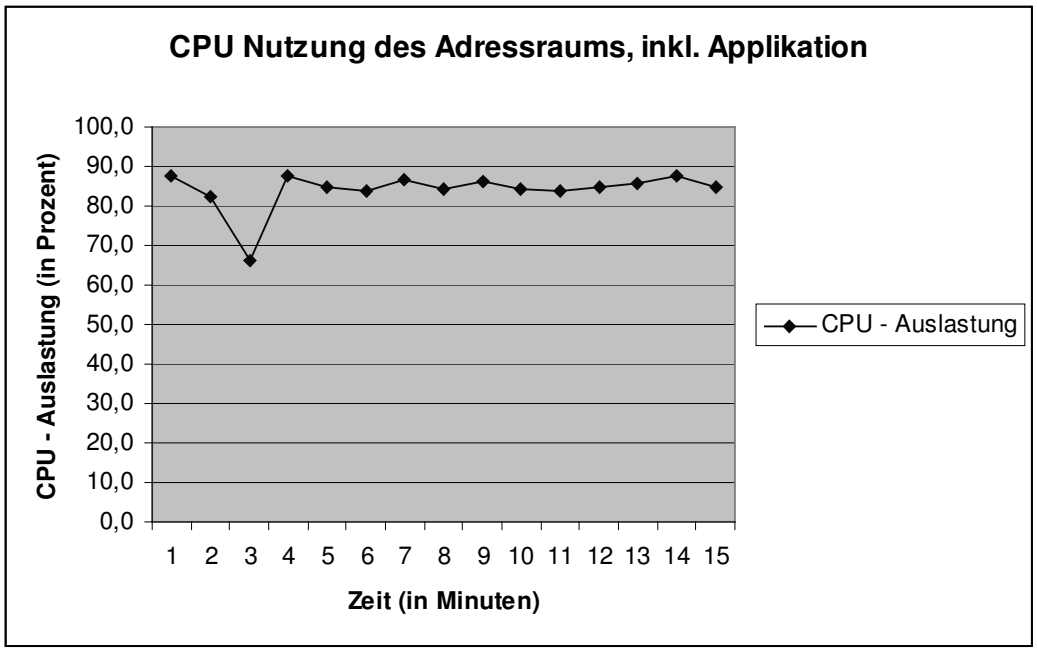

Abbildung 57: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 03

Mittelwert	84,0
Standardabweichung	5,21
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	6,3 %
Signifikanz	Marginal signifikant

Tabelle 38: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 03

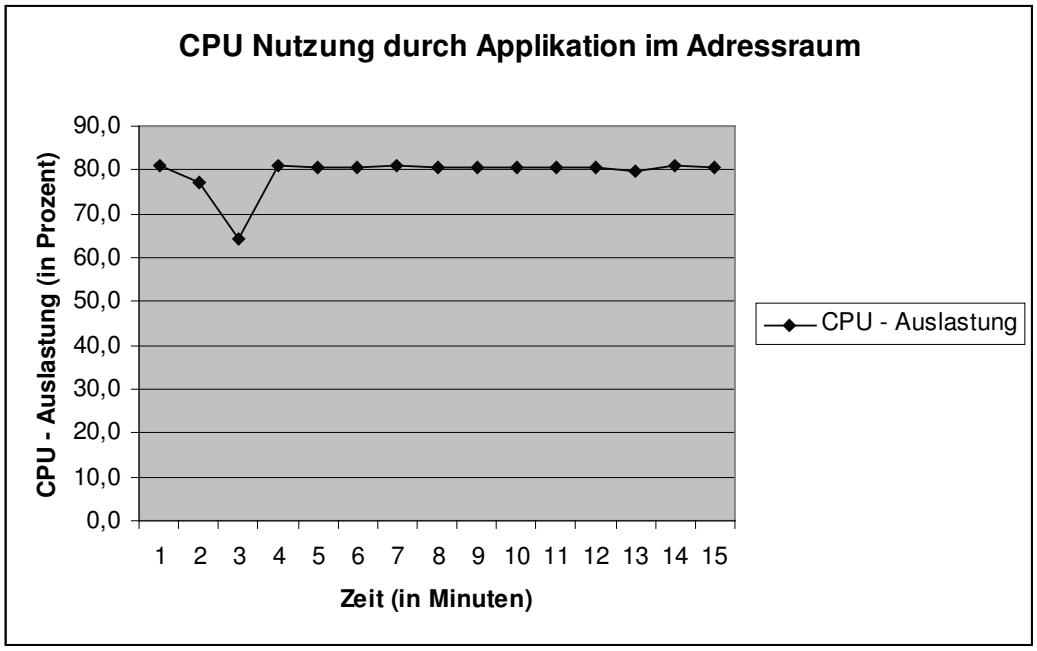

Abbildung 58: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 03

Mittelwert	79,3
Standardabweichung	4,27
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	6,3 %
Signifikanz	Marginal signifikant

Tabelle 39: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 03

13.7.2.2.4 Performance-Messung Nr. 04: SWF #003 – Sequentiell, mit Alternative (Java Aktivität)

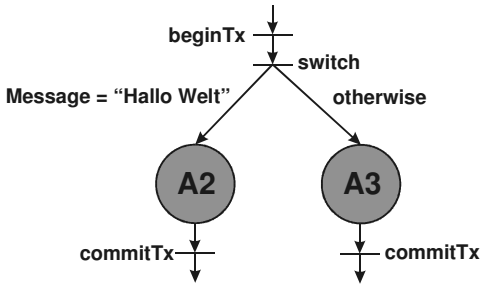

Abbildung 59: SWF #003: Sequentiell, mit Alternative (Java Aktivität)

Die Aktivität A3 prüft, ob der Message String „Hallo Welt“ ist. Wenn ja, dann gibt die Aktivität A4 über das Notification API „Message = ‚Hallo Welt’“ aus und andernfalls gibt die Aktivität A5 „Message != ‚Hallo Welt’“ über das Notification API aus.

Transaktionen	100.000
Transaktionen pro Sekunde	97,8

Tabelle 40: Überblick über die Performance-Messung Nr. 04

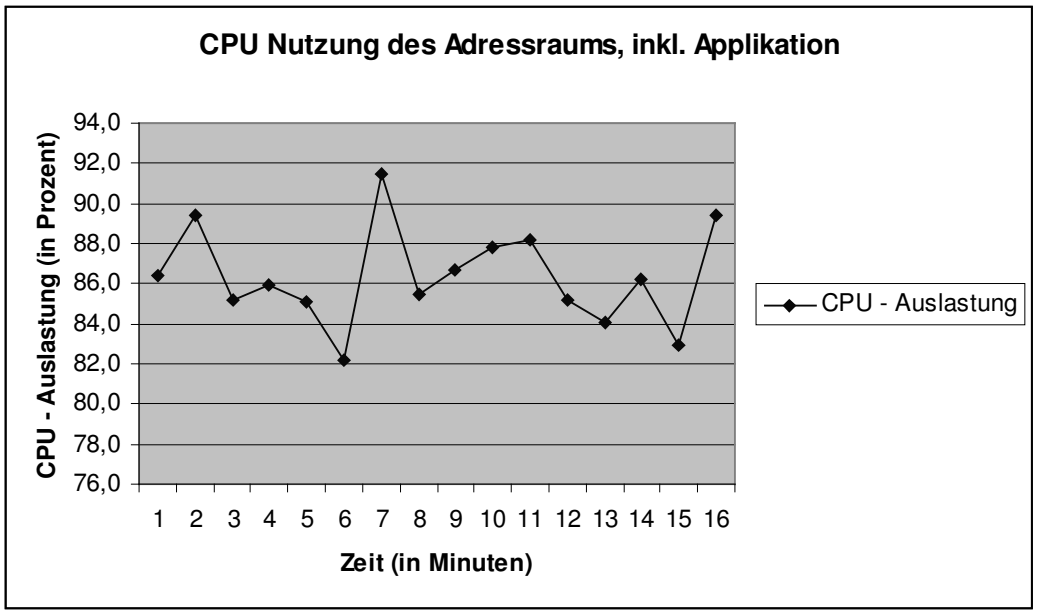

Abbildung 60: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 04

Mittelwert	86,4
Standardabweichung	2,45
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	6,3 %
Signifikanz	Marginal signifikant

Tabelle 41: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 04

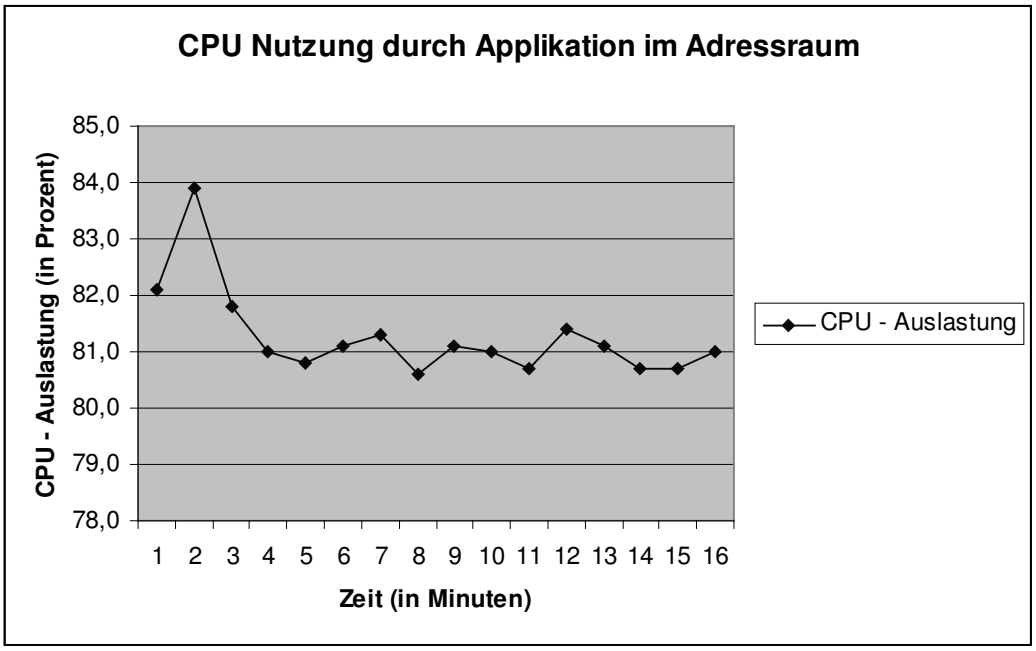

Abbildung 61: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 04

Mittelwert	81,3
Standardabweichung	0,81
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	6,3 %
Signifikanz	Marginal signifikant

Tabelle 42: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 04

13.7.2.2.5 Performance-Messung Nr. 05: SWF #004 – Iterativ sequentiell (Java Aktivität)

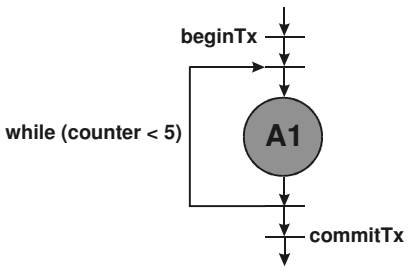

Abbildung 62: SWF #004: Iterativ sequentiell (Java Aktivität)

Die counter Variable wird zu Beginn in der DataSectionCopystruktur initialisiert (=0) und anschließend wird die Aktivität fünfmal ausgeführt, d.h. es wird fünfmal der Message String über das Notification API ausgegeben.

Transaktionen	100.000
Transaktionen pro Sekunde	72,7

Tabelle 43: Überblick über die Performance-Messung Nr. 05

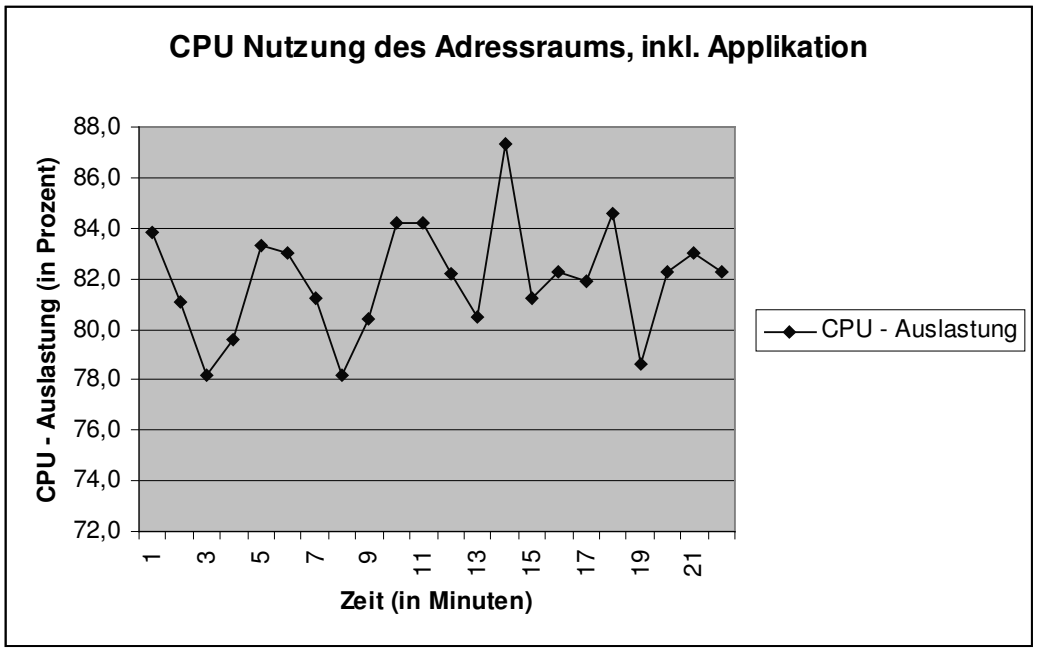

Abbildung 63: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 05

Mittelwert	82,0
Standardabweichung	2,23
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	4,5 %
Signifikanz	Signifikant

Tabelle 44: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 05

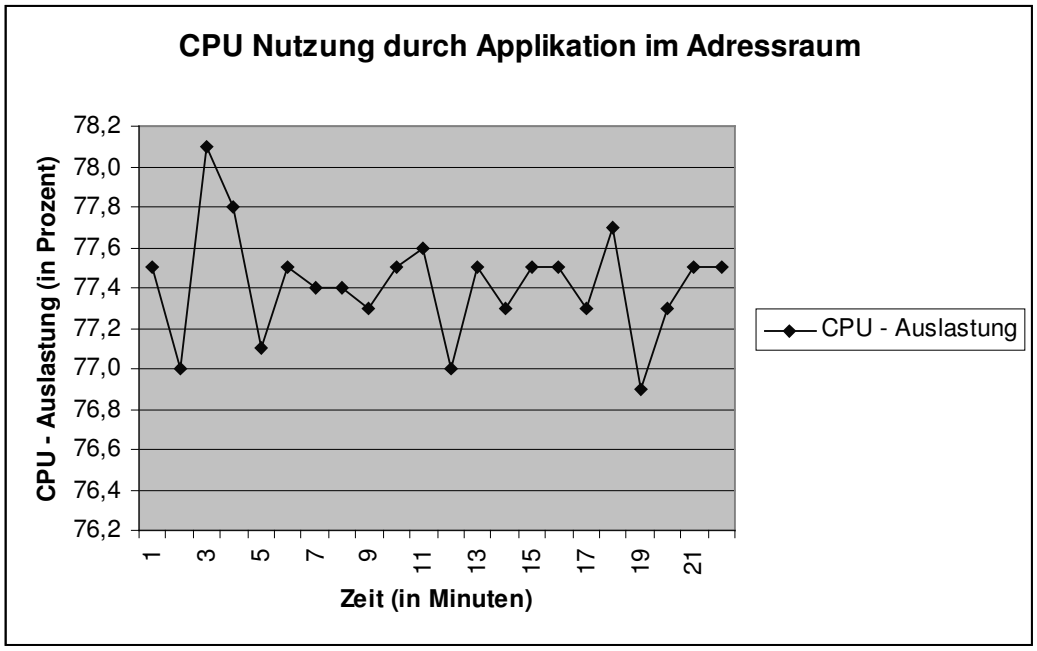

Abbildung 64: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 05

Mittelwert	77,4
Standardabweichung	0,27
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	4,5 %
Signifikanz	Signifikant

Tabelle 45: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 05

13.7.2.2.6 Performance-Messung Nr. 06: SWF #005 – Sequentiell ohne Fehler (COBOL Aktivität)

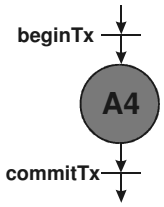

Abbildung 65: SWF #005: Sequentiell ohne Fehler (COBOL Aktivität)

Der SWF ist vom Ablauf analog zu SWF #001: der Message String wird ausgegeben, wobei hier der eigentliche COBOL Aufruf dadurch simuliert wird, dass eine C Bridge aufgerufen wird, die wiederum eine C Methode aufruft, die dann aus dem Java COBOL Objekt den Message String „liest“ und ausgibt (printf).

Transaktionen	100.000
Transaktionen pro Sekunde	101,2

Tabelle 46: Überblick über die Performance-Messung Nr. 06

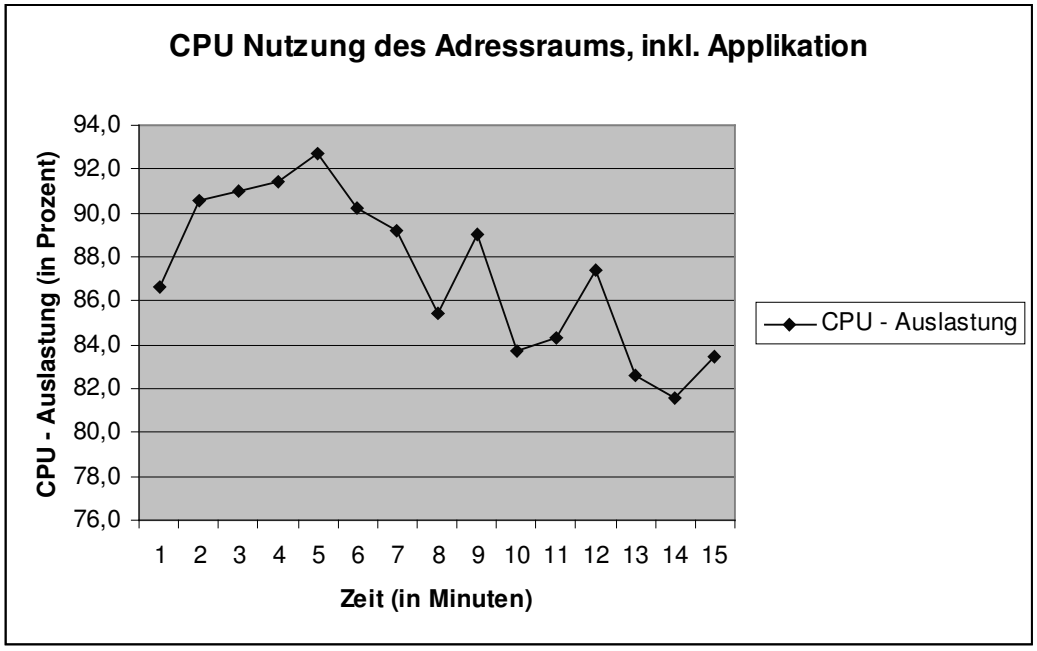

Abbildung 66: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 06

Mittelwert	87,3
Standardabweichung	3,59
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 47: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 06

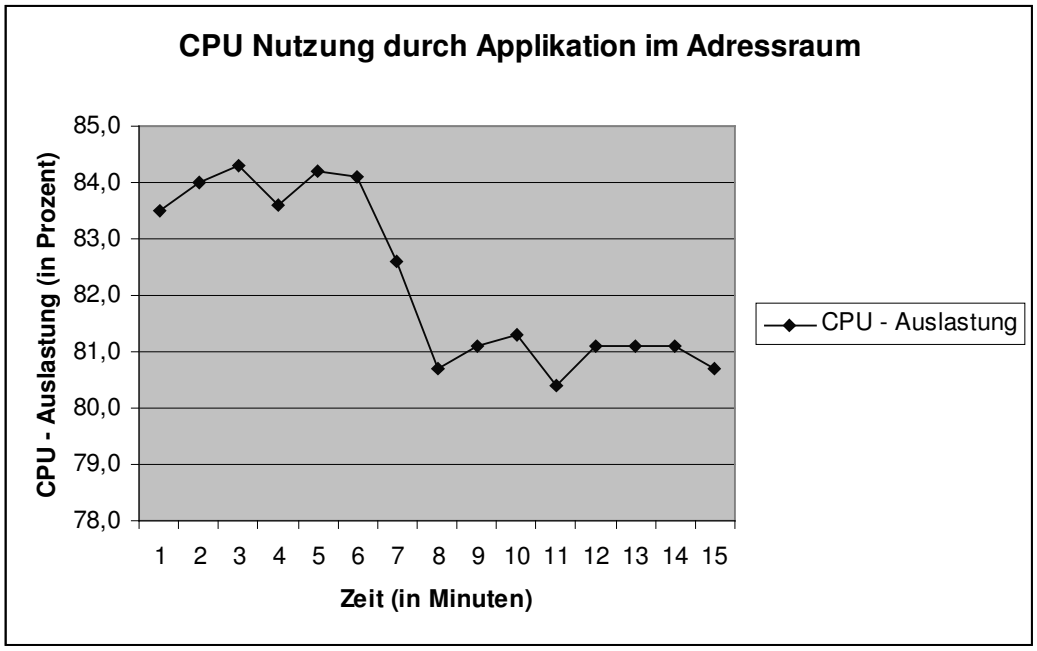

Abbildung 67: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 06

Mittelwert	82,3
Standardabweichung	1,52
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 48: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 06

13.7.2.3 TPC A Vergleich mit Referenz Messung

13.7.2.3.1 Überblick

accountID	tellerID	branchID	delta

Abbildung 68: Globale Eingabedaten für SWF #006

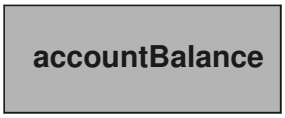

Abbildung 69: Globale Rückgabedaten für SWF #006

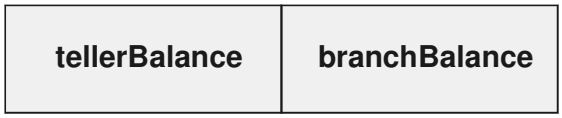

Abbildung 70: Lokale Daten für SWF #006

Zu Beginn des SWF werden wie in [TPC] spezifiziert aus drei verschiedenen Tabellen einer Datenbank (ACCOUNTS, TELLERS, BRANCHES) die Werte für accountBalance, tellerBalance und branchBalance initialisiert.

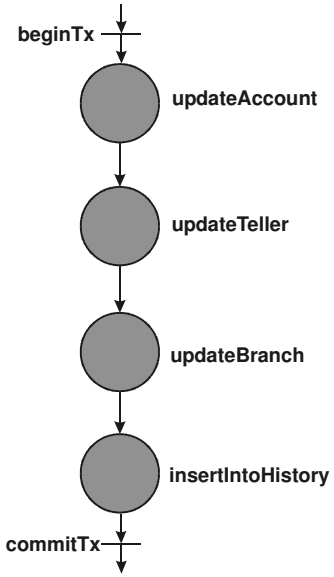

Abbildung 71: SWF #006: TPC A

Der SWF besteht aus vier Schritten: zuerst wird die accountBalance um den Wert delta erhöht (delta kann auch negativ sein) (updateAccount), anschließend wird die tellerBalance um den Wert delta erhöht (updateTeller), danach wird die branchBalance um den Wert delta erhöht (updateBranch). Zum Schluss wird eine Verwaltungsinformation (Log) in die Datenbank geschrieben, die aus accountID, tellerID, branchID, delta und dem augenblicklichen Timestamp besteht (insertIntoHistory). Sowohl accountBalance, tellerBalance, branchBalance als auch delta sind nach Vorgabe des TPC A Beispiels vom Typ long.

Da dieser SWF als Vergleich mit der in Kapitel 13.7.2.1 beschriebenen Referenz Messung dient, wurden hier fünf verschiedene Szenarien gemessen, bei denen folgende Performance-Parameter berücksichtigt wurden:

- Caching von Methoden an / aus: Wie in Kapitel 2.2: Persistent Reusable Java Virtual Machine (PRJVM) angeführt, darf keine Middleware Klasse eine Referenz auf eine Applikationsklasse zum Zeitpunkt des Zurücksetzens (reset) auf einen „sauberen Systemzustand“ halten. Daraus folgt, dass bei jeder Ausführung eines Subworkflows, die zugehörige Applikationsklasse erneut per Java Reflection (siehe [ECK] und den auf der beigelegten CD enthalten Source Code) bestimmt werden muss. Um den damit verbundenen Performance-Overhead bestimmen zu können, wurde eine alternative Implementierung entwickelt, die es ermöglicht diese Referenzen auf Kosten der Isolationseigenschaft zwischenzuspeichern (cachen).
- Garbage Collection an / aus: Die Persistent Reusable Java Virtual Machine (PRJVM, siehe [JVM]) ermöglicht es, an „beliebigen“ Stellen die Garbage Collection (siehe [ARN]) von Java anzustoßen. Nachteilig ist hier allerdings, dass dieser Wert manuell bestimmt und einmalig für alle Ausführungen gesetzt werden muss. Um zu sehen, wie viel Performance-Overhead durch einen suboptimalen Garbage Collection Zeitpunkt entsteht, wurde versucht einen Durchlauf ohne Garbage Collection zu messen.
- Hoher / niedriger Isolationslevel: JDBC bzw. die eingesetzte Datenbank (DB2) unterstützt unterschiedliche Isolationslevel beim Zugriff auf die Datenbank (siehe [FIS]). Je restriktiver die Zugriffskontrolle, desto mehr Locks werden bei jedem Zugriff gehalten und desto langsamer ist der Zugriff, bzw. je restriktiver die Zugriffskontrolle, desto weniger Nebeneffekte, wie z.B. nicht-wiederholbare Datenzugriffe (non-repeatable reads). Um zu bestimmen, wie viel Performance-Overhead durch die gehaltenen Locks entsteht, wurden bei der Messung unterschiedliche Werte für diesen Parameter gewählt.
- Java Test Treiber: Nachdem sich bei ersten Messungen Latenzzeiten beim Zugriff auf den in [BEY] beschriebenen angepassten C Launcher, der als Carrier (und somit auch als In und Out Queue) fungiert, zeigten, wurde ein alternativer Java Test Treiber entwickelt. Dieser Java Test Treiber simuliert einen primitiven Carrier (und somit die In und Out Queue) und läuft direkt unter z/OS, so dass die Trennung der Messung von Test Treiber und Subworkflowsteuerung in dieser Variante nicht mehr gegeben ist.

Anmerkung zur Implementierung: Um das Einfügen von neuen Reihen in Datenbanken zu ermöglichen wurde die lokale Sicht auf den Message Container und der Data Service um die Möglichkeit neue Reihen in eine bestehende Tabelle einzufügen erweitert.

13.7.2.3.2Performance-Messung Nr. 07: SWF #006 – TPC A; Szenario 1: Caching von Methoden aus (Normalbetrieb)

Transaktionen	100.000
Transaktionen pro Sekunde	24,6

Tabelle 49: Überblick über die Performance-Messung Nr. 07

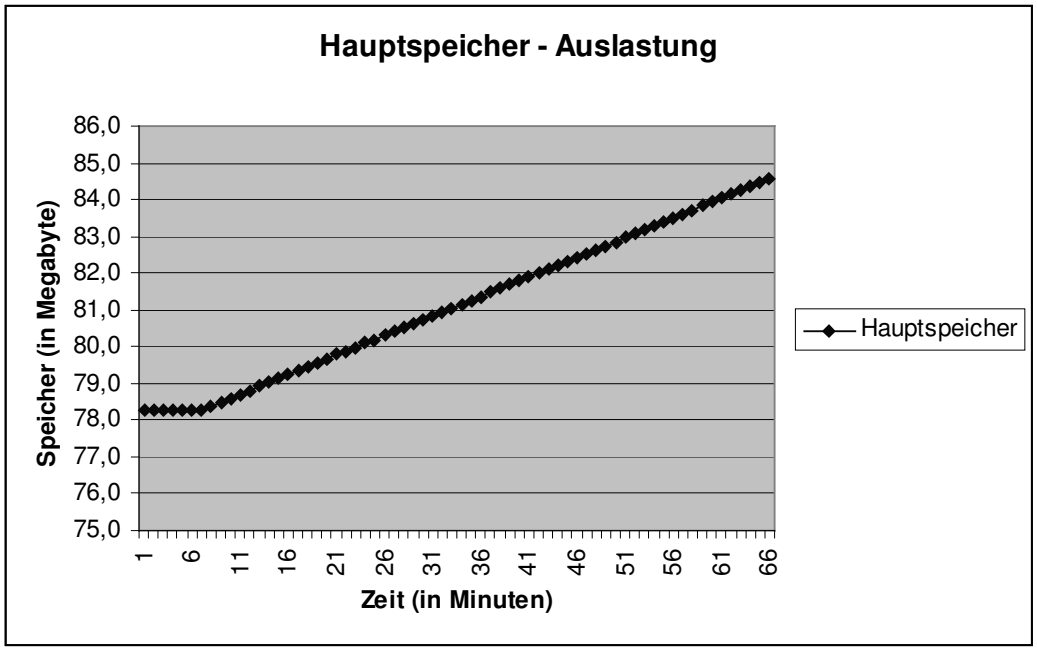

Abbildung 72: Hauptspeicher-Auslastung für Performance-Messung Nr. 07

Mittelwert	81,1
Standardabweichung	1,99
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 50: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 07

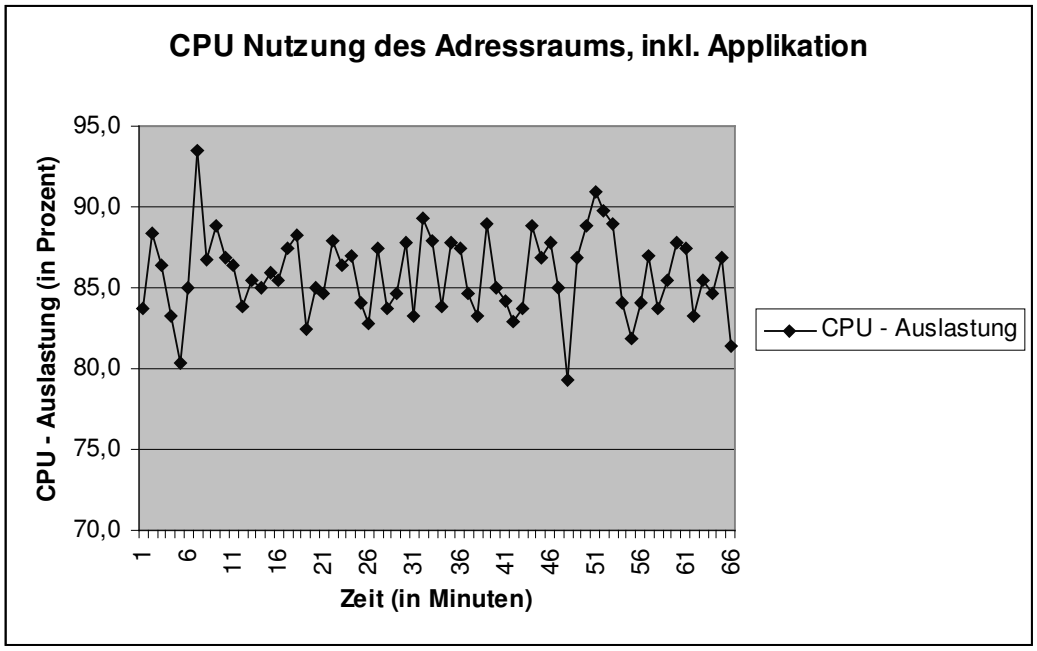

Abbildung 73: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 07

Mittelwert	85,8
Standardabweichung	2,56
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	3,0 %
Signifikanz	Signifikant

Tabelle 51: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 07

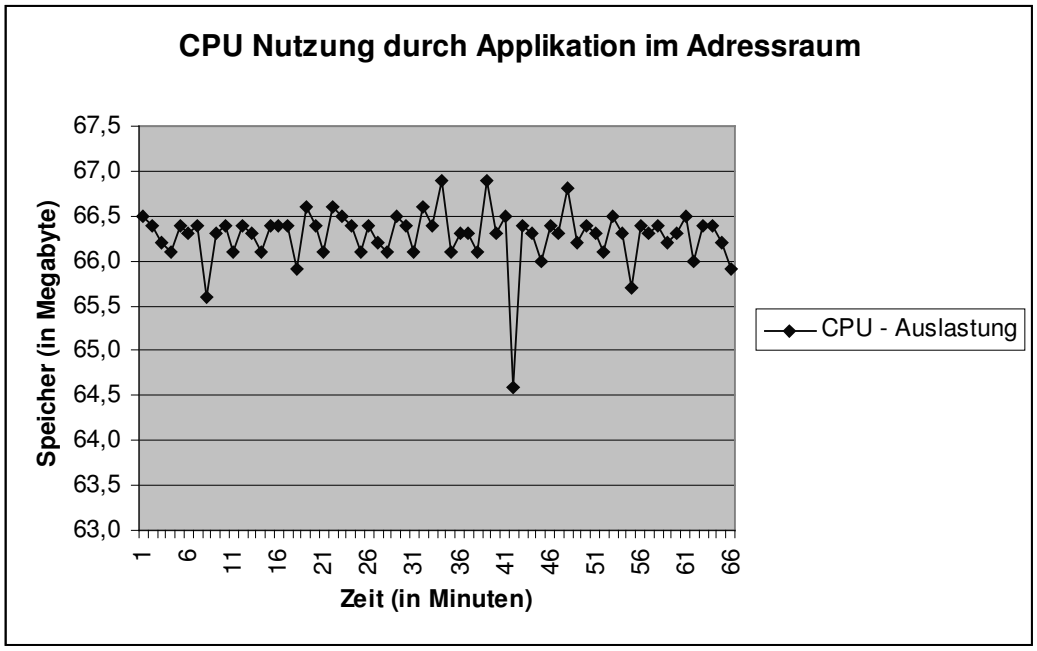

Abbildung 74: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 07

Mittelwert	66,3
Standardabweichung	0,31
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	3,0 %
Signifikanz	Signifikant

Tabelle 52: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 07

13.7.2.3.3 Performance-Messung Nr. 08: SWF #006 – TPC A; Szenario 2: Caching von Methoden ein

Transaktionen	40.000
Transaktionen pro Sekunde	24,7

Tabelle 53: Überblick über die Performance-Messung Nr. 08

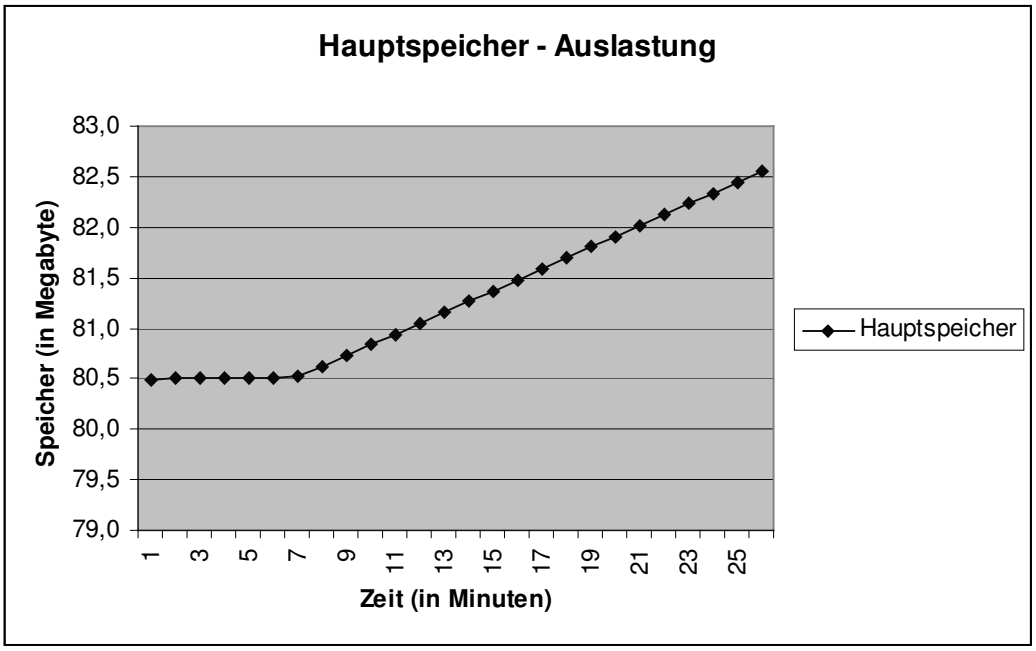

Abbildung 75: Hauptspeicher-Auslastung für Performance-Messung Nr. 08

Mittelwert	81,3
Standardabweichung	0,71
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 54: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 08

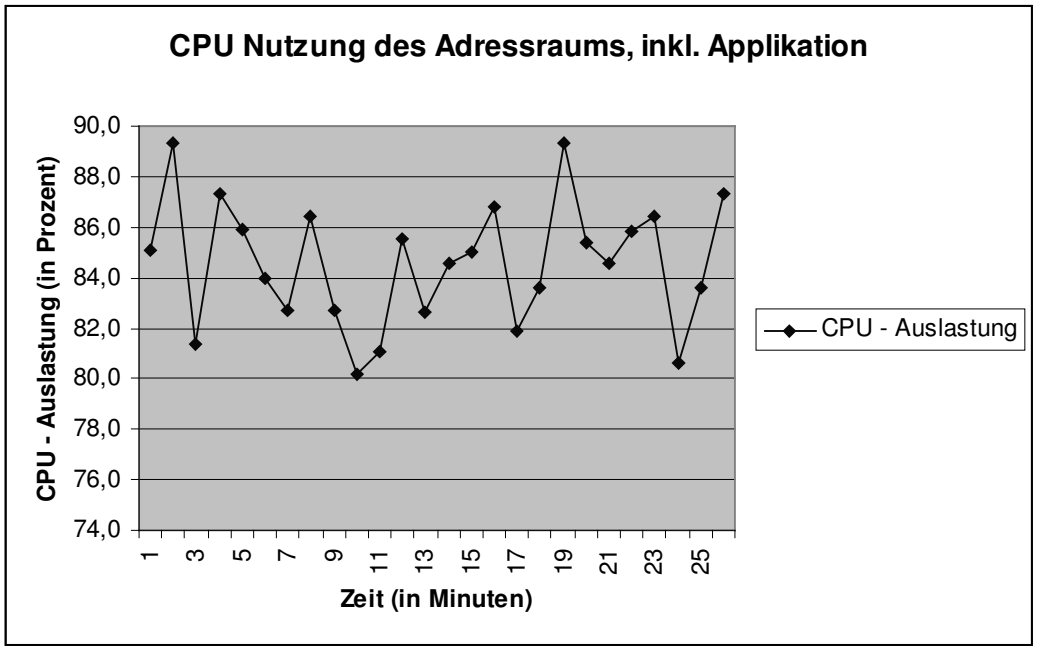

Abbildung 76: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 08

Mittelwert	84,6
Standardabweichung	2,48
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 55: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 08

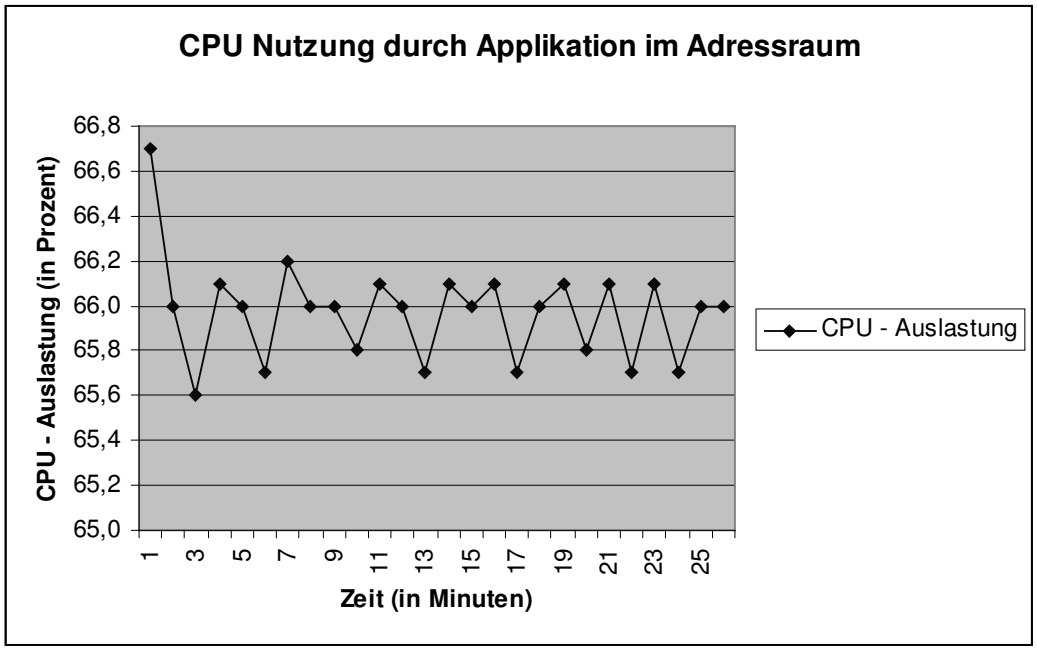

Abbildung 77: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 08

Mittelwert	66,0
Standardabweichung	0,23
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	3,8 %
Signifikanz	Signifikant

Tabelle 56: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 08

13.7.2.3.4 Performance-Messung Nr. 09: SWF #006 – TPC A; Szenario 3: Caching von Methoden ein, Garbage Collection aus

Transaktionen	40.000
Transaktionen pro Sekunde	19,1

Tabelle 57: Überblick über die Performance-Messung Nr. 09

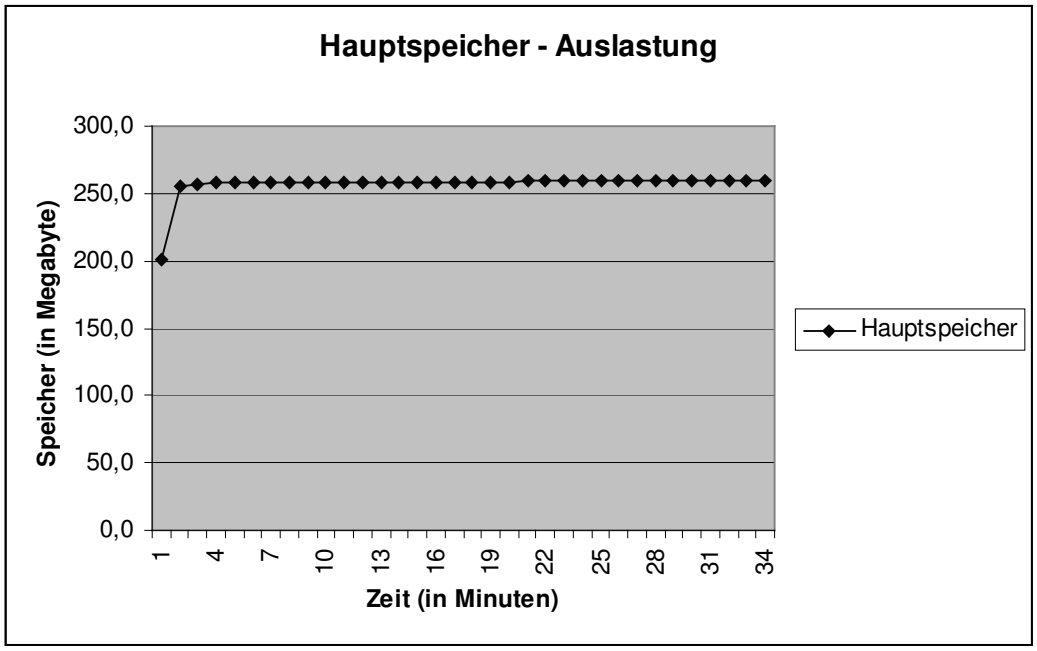

Abbildung 78: Hauptspeicher-Auslastung für Performance-Messung Nr. 09

Mittelwert	256,9
Standardabweichung	9,95
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	2,9 %
Signifikanz	Signifikant

Tabelle 58: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 09

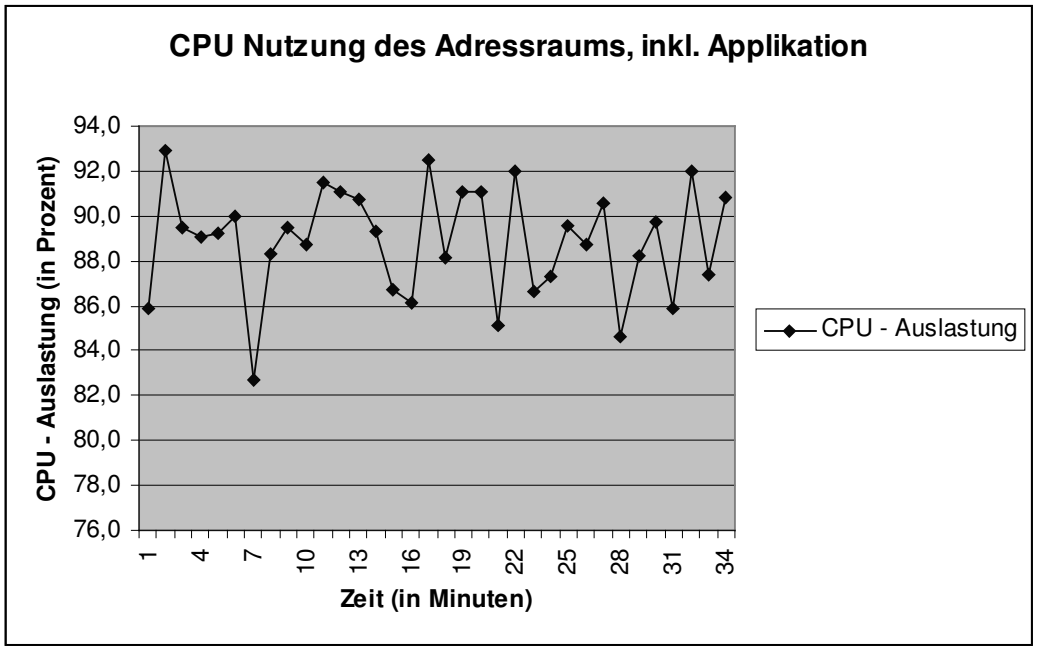

Abbildung 79: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 09

Mittelwert	88,9
Standardabweichung	2,44
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	2,9 %
Signifikanz	Signifikant

Tabelle 59: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 09

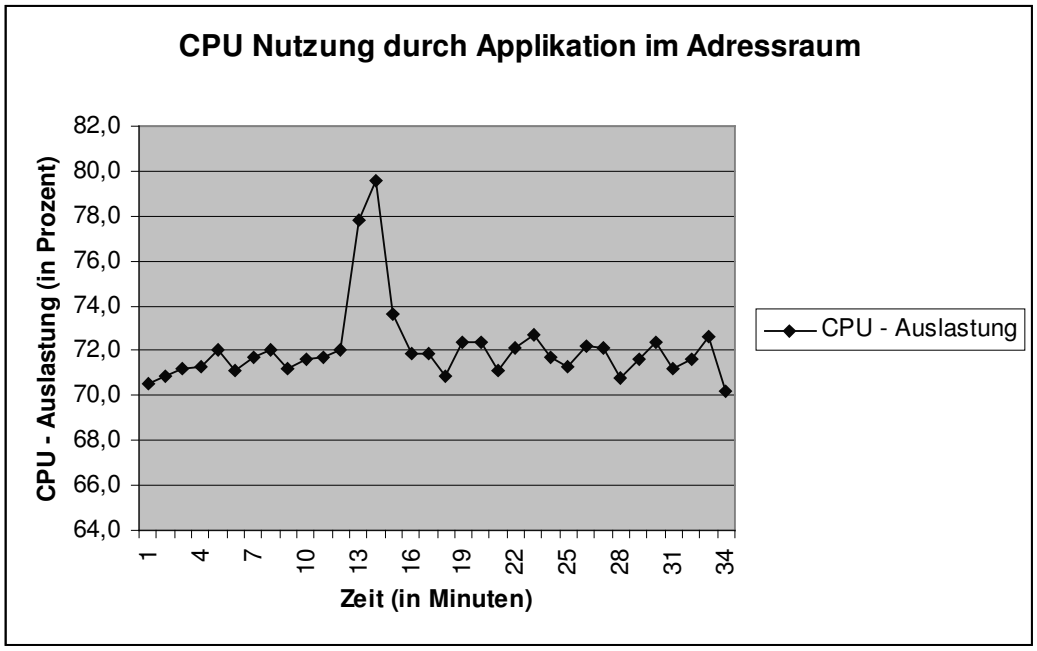

Abbildung 80: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 09

Mittelwert	72,1
Standardabweichung	1,82
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,9 %
Signifikanz	Marginal signifikant

Tabelle 60: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 09

13.7.2.3.5Performance-Messung Nr. 10: SWF #006 – TPC A; Szenario 4: Caching von Methoden ein, niedriger Isolationslevel

Transaktionen	40.000
Transaktionen pro Sekunde	24,7

Tabelle 61: Überblick über die Performance-Messung Nr. 10

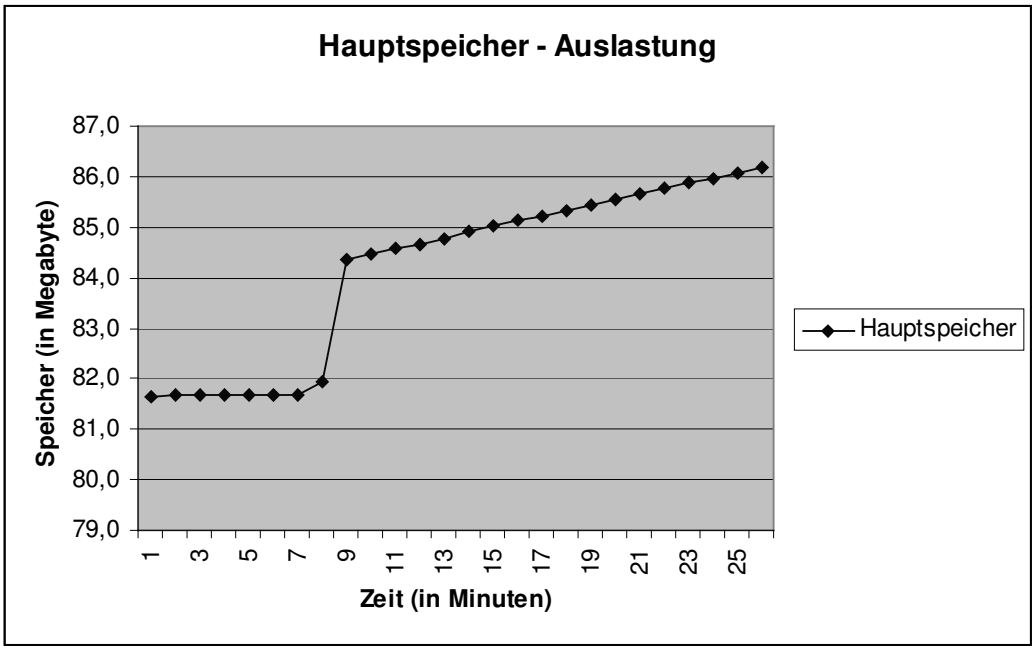

Abbildung 81: Hauptspeicher-Auslastung für Performance-Messung Nr. 10

Mittelwert	84,2
Standardabweichung	1,75
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 62: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 10

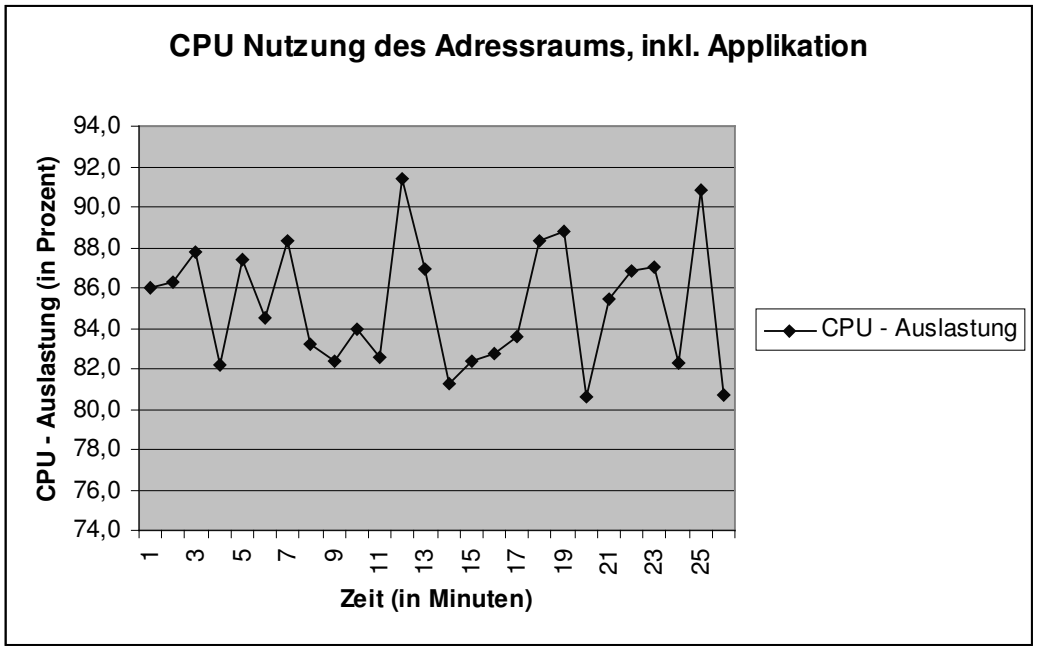

Abbildung 82: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 10

Mittelwert	85,1
Standardabweichung	3,07
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	3,8 %
Signifikanz	Signifikant

Tabelle 63: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 10

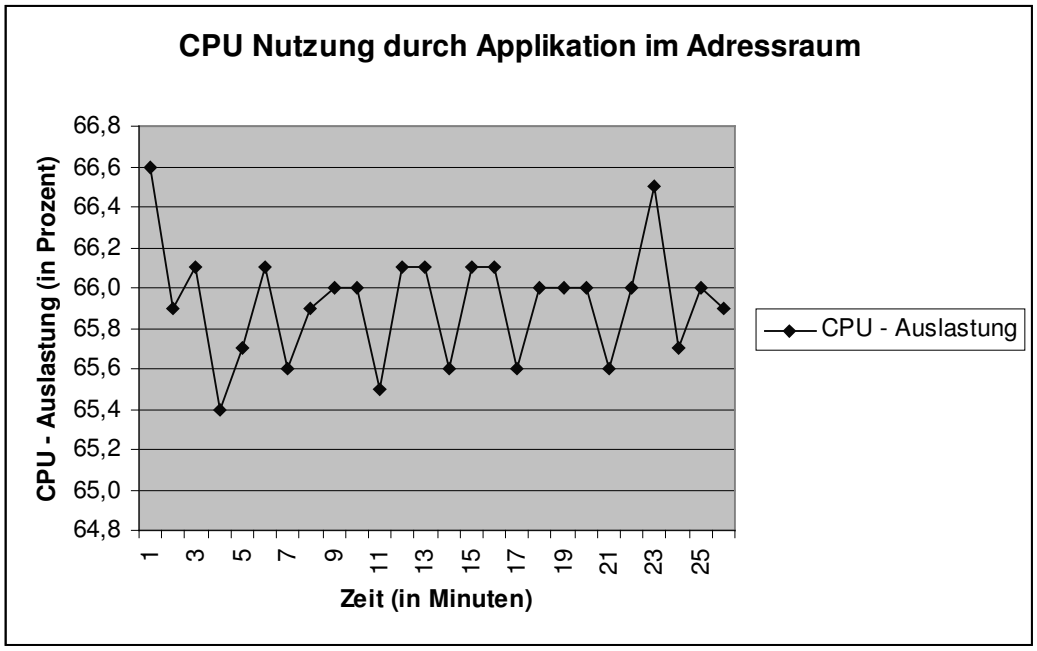

Abbildung 83: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 10

Mittelwert	65,9
Standardabweichung	0,28
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	7,7 %
Signifikanz	Marginal signifikant

Tabelle 64: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 10

13.7.2.3.6 Performance-Messung Nr. 11: SWF #006 – TPC A; Szenario 5: Caching von Methoden ein, niedriger Isolationslevel, Java Test Treiber

Transaktionen	20.000
Transaktionen pro Sekunde	28,2

Tabelle 65: Überblick über die Performance-Messung Nr. 11

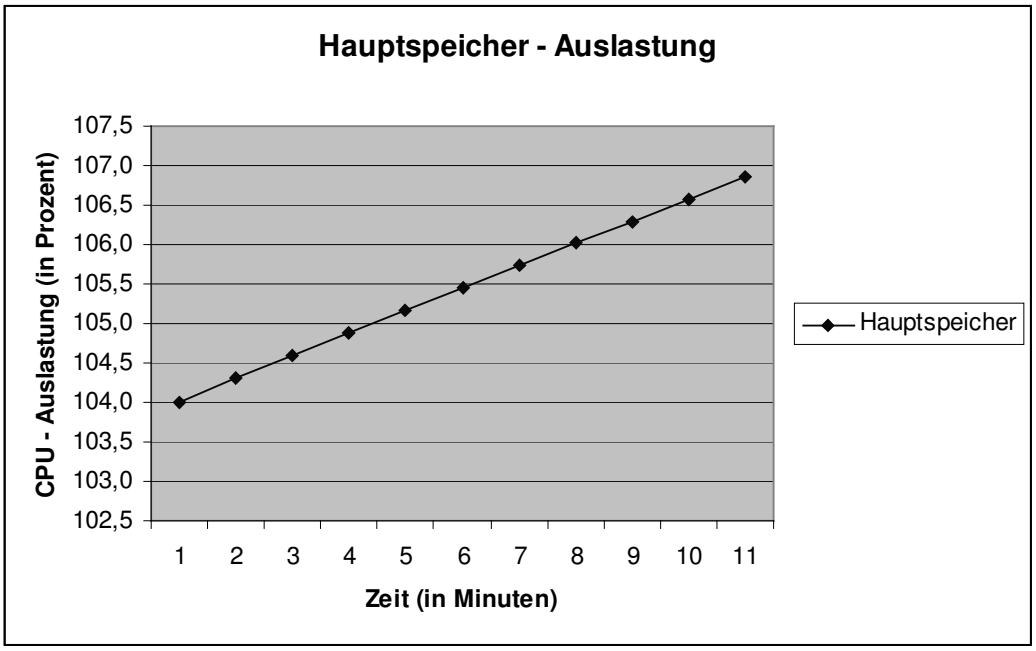

Abbildung 84: Hauptspeicher-Auslastung für Performance-Messung Nr. 11

Mittelwert	105,4
Standardabweichung	0,94
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 66: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 11

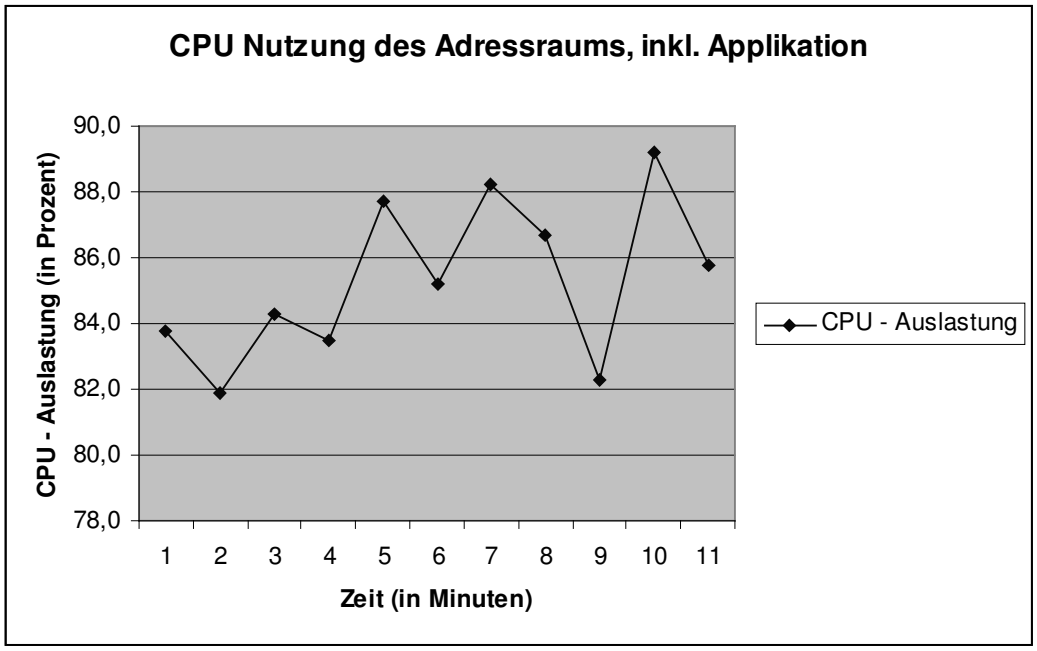

Abbildung 85: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 11

Mittelwert	85,3
Standardabweichung	2,42
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 67: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 11

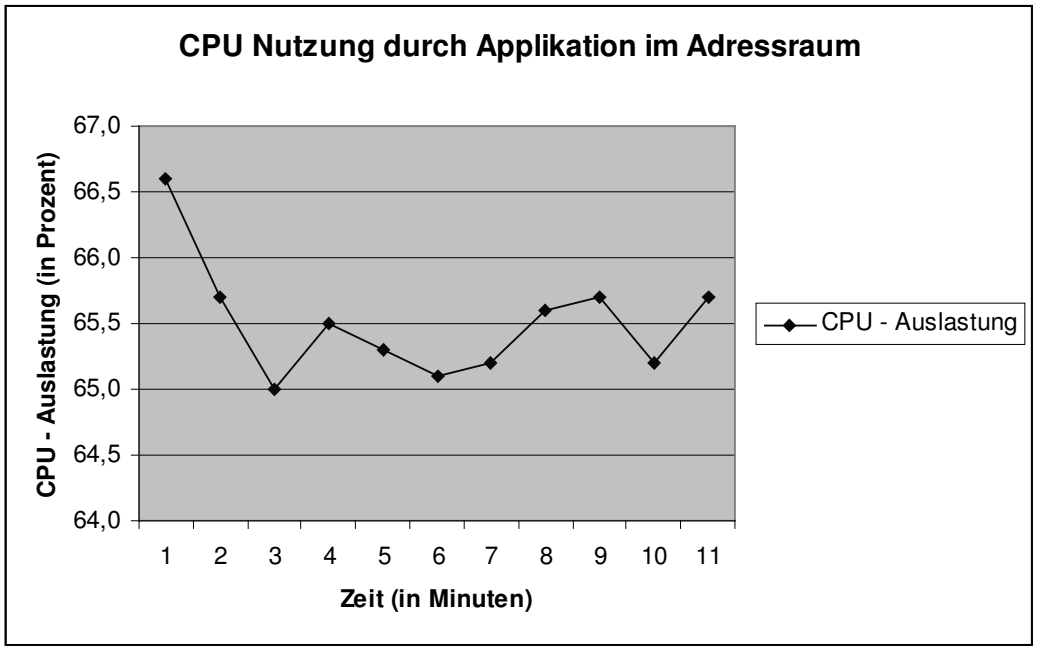

Abbildung 86: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 11

Mittelwert	65,5
Standardabweichung	0,44
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	9,1 %
Signifikanz	Marginal signifikant

Tabelle 68: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 11

13.7.2.4 Einkauf in einem Internet Shop

13.7.2.4.1 Überblick

Dieser Subworkflow modelliert einen Einkauf in einem Internet Shop. Hierbei stehen drei verschieden Shops zur Wahl: Shop A, Shop B und Shop C. Bei allen drei Shops gibt es die Möglichkeit Bonuspunkte für einen Einkauf zu sammeln, die sich anhand der Höhe der Einkaufssumme berechnen. Übersteigen die Bonuspunkte eine gewisse Summe (100.000), dann bekommt der Kunde einen Rabatt von 10 % auf den Kaufpreis bei jedem weiteren Kauf. Der SWF ist unterteilt in drei Schritte: zuerst wird die Anzahl der zusätzlichen Bonuspunkte errechnet, dann erfolgt die Zahlung (debit-credit) und zum Schluss wird geprüft, ob die Summe der Bonuspunkte größer als 100.000 ist. Ist dies der Fall, bekommt der Kunde 10 % des Kaufpreises zurückerstattet (wobei hier

sowohl debit als auch credit in einer Aktivität gekapselt werden). Die zu diesem Szenario gehörige Datenbank besteht aus den Tabellen Tabelle 69 und Tabelle 70. Die hier angegeben Werte entsprechen den Initialwerten der Felder.

CUSTOMERS

ACCOUNTNO	BALANCE	BONUSPOINTS
1234567890	1,000,000.00	0
2345678901	1,000,000.00	0
3456789012	1,000,000.00	0
4567890123	1,000,000.00	0

Tabelle 69: SWF #007-SWF #011: Datenbanktabelle CUSTOMERS

SHOPS

SHOPNAME	BALANCE
Shop A	0.0
Shop B	0.0
Shop C	0.0

Tabelle 70 SWF #007-SWF #011: Datenbanktabelle SHOPS

Der hier beschriebene Geschäftsprozess wurde in sechs verschiedenen Varianten gemessen:

- Variante 1: der Geschäftsprozess wurde implementiert als ein Subworkflow bestehend aus drei Transaktionen
- Variante 2: der Geschäftsprozess wurde unterteilt in drei verschiedene Subworkflows, die den drei Transaktionen von Variante 1 entsprechen. Diese Implementierung entspricht einer Stapelverarbeitung (Batch job). Zuerst werden für alle Einkäufe die Bonuspunkte berechnet, dann alle Zahlungen durchgeführt und zum Schluss die eventuellen Rücküberweisungen getätigt. Hierbei entstehen andere Schlusskontostände als in Variante 1, da gegeben durch die Dreiteilung des Geschäftsprozesses alle Kunden eine Rücküberweisung schon beim ersten Kauf bekommen.
- Variante 3: der Geschäftsprozess wurde ähnlich durchgeführt wie Variante 2, jedoch mit dem Unterschied, dass zuerst ein Subworkflow ausgeführt wurde, der die Bonuspunkte berechnet, danach der zugehörige Subworkflow, der die Zahlung

durchführt und zum Schluss der zugehörige Subworkflow, der die Rücküberweisung vornimmt – dadurch ist der Schluss-Kontostand bei allen Konten genau wie in Variante 1. Diese Messung wurde durchgeführt, um bewerten zu können, wie viel Performance-Gewinn sich durch die Verkettung von mehreren Transaktionen in einem Subworkflow realisieren lässt.

- Variante 4: der Geschäftsprozess wurde implementiert als ein Subworkflow bestehend aus einer Transaktion. Diese Messung dient dazu, ein theoretisches Maximum für die Zeitersparnis des Data Service zu finden, da dieser den Zeitaufwand für die zusätzlichen Rückschreibungen der geänderten globalen und lokalen Daten bei den zwei zusätzlichen commits bei Variante 1 (vergleiche Abbildung 89 und Abbildung 116) im Gegensatz zu einem „realen“ Zugriff auf die Datenbank minimieren sollte.
- Variante 5: der Geschäftsprozess wurde durchgeführt, wie in Variante 1, jedoch mit dem Unterschied, dass 25 % der credit-Aktivitäten mit dem Return Code ROLLBACKTX enden.
- Variante 6: der Geschäftsprozess wurde durchgeführt, wie in Variante 1, jedoch mit dem Unterschied, dass 50 % der credit-Aktivitäten mit dem Return Code ROLLBACKTX enden. Die Varianten 4 und 5 wurden gemessen, um den Performance-Overhead durch de Fehlerbehandlung der Subworkflowsteuerung bewerten zu können.

13.7.2.4.2Performance-Messung Nr. 12: Einkauf in einem Internet Shop; Variante 1: SWF #007 – ein SWF bestehend aus drei Transaktionen

amount	accountNo	shopName	rollbackTx

Abbildung 87: Globale Daten für SWF #007

bonusPoints	balanceCustomer	balanceShop

Abbildung 88: Lokale Daten für SWF #007

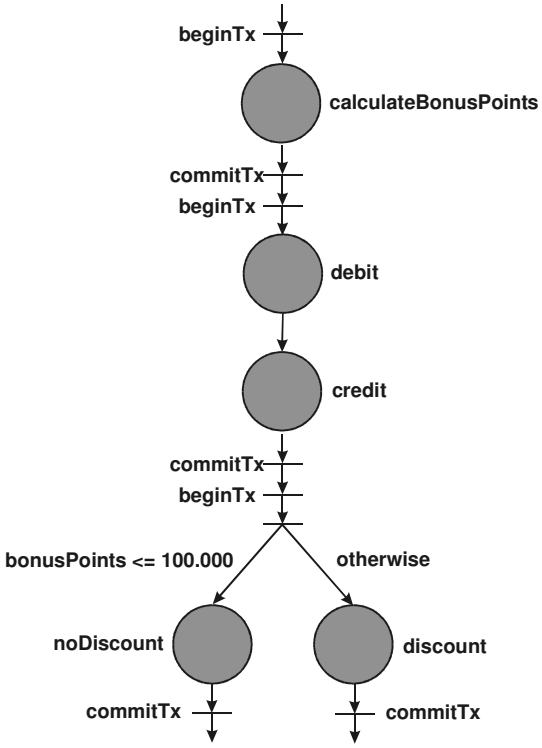

Abbildung 89: SWF #007: ein SWF bestehend aus drei Transaktionen

Zu Beginn des SWF werden die globalen Werte balanceCustomer, balanceShop und der lokale Wert bonusPoints mit Hilfe der Datenbank initialisiert.

Über die den globalen Wert rollbackTx (siehe Abbildung 87) in der SWFC RWU kann spezifiziert werden, ob die credit-Aktivität mit dem Return Code RC_ROLLBACKTX enden soll. Dies wird in in Kapitel 13.7.2.4.6 eingesetzt, um die Abbruchszenarien einfach realisieren zu können.

Wenn dem globalen Datum rollbackTx in der SWFC RWU der Wert „true“ zugewiesen ist, führt der SWF Interpreter einen rollbackTx durch und startet die Ausführung des SWF wieder mit der Aktivität debit im Ausführungskontext rollbackTx. Die Aktivität debit überprüft, ob der Ausführungskontext rollbackTx vorliegt, und setzt gegebenenfalls den globalen Wert rollbackTx auf den Wert „false“, so dass die Aktivität credit beim nächsten Versuch ausgeführt wird. Der Wert rollbackTx dient somit nur dazu, um die in 13.7.2.4.6 dargestellten Abbruch Szenarien komfortabel zu realisieren und hat bei dieser Variante immer den Wert „true“.

Transaktionen	100.000
Transaktionen pro Sekunde	15.5

Tabelle 71: Überblick über die Performance-Messung Nr. 12

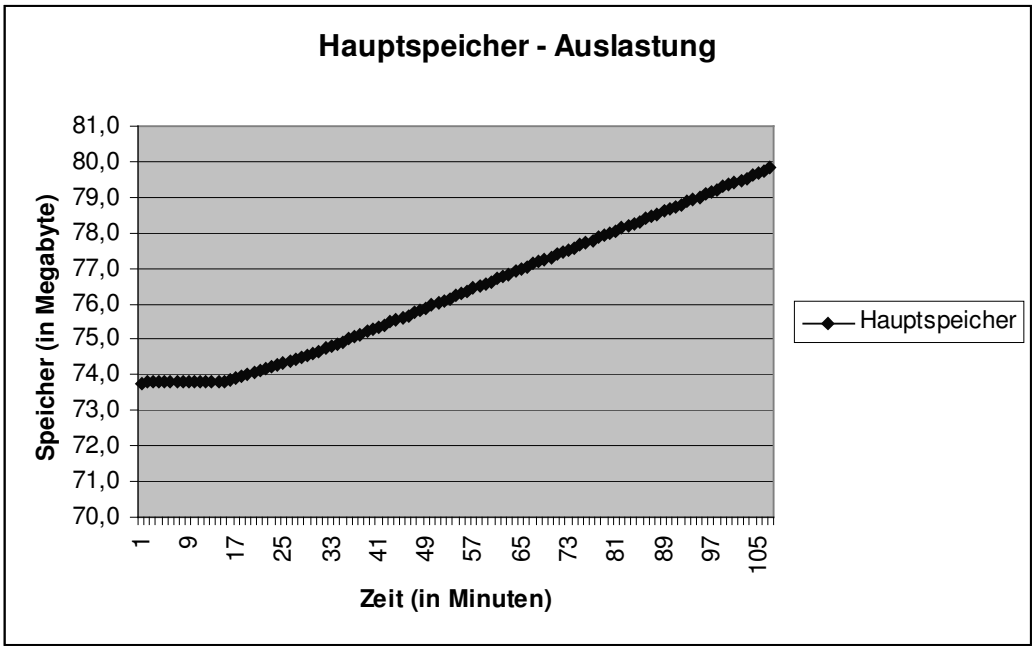

Abbildung 90: Hauptspeicher-Auslastung für Performance-Messung Nr. 12

Mittelwert	76,3
Standardabweichung	1,95
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 72: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 12

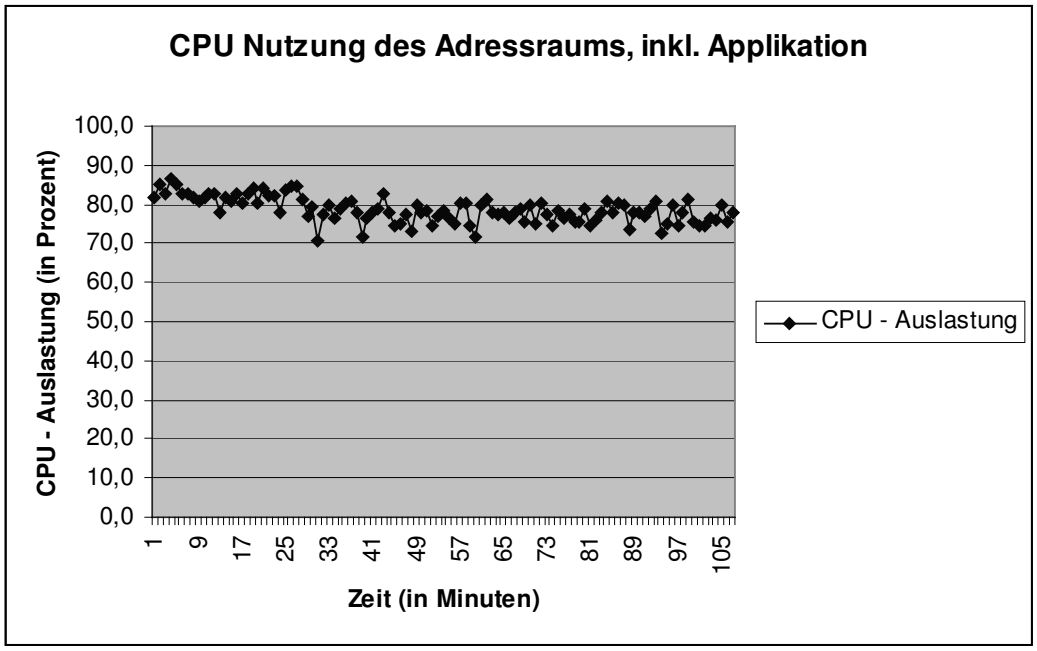

Abbildung 91: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 12

Mittelwert	78,6
Standardabweichung	3,28
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,6 %
Signifikanz	Marginal signifikant

Tabelle 73: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 12

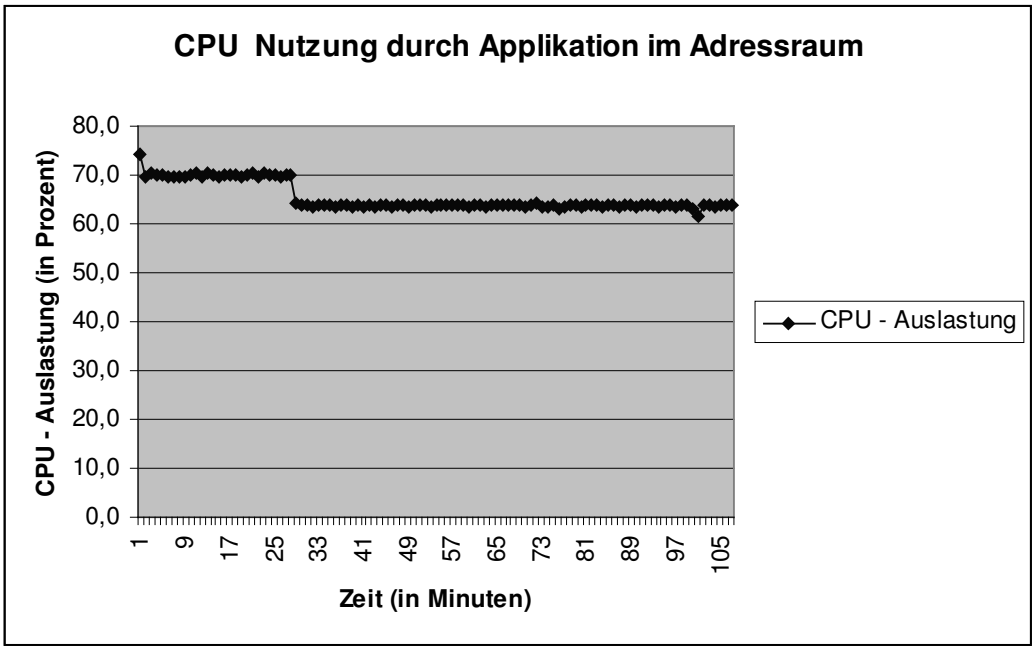

Abbildung 92: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 12

Mittelwert	65,4
Standardabweichung	2,84
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,9 %
Signifikanz	Sehr signifikant

Tabelle 74: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 12

13.7.2.4.3Performance-Messung Nr. 13-15: Einkauf in einem Internet Shop; Variante 2: SWF #008 bis SWF #010 – drei Subworkflows bestehend aus je einer Transaktion

amount	**accountNo**

Abbildung 93: Globale Daten für SWF #008

Abbildung 94: Lokale Daten für SWF #008

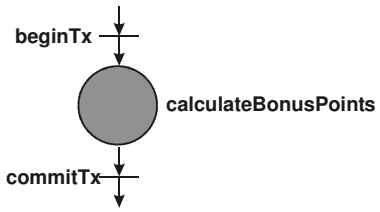

Abbildung 95: SWF #008: Erster Teil des SWF #007

Zu Beginn des SWF wird der lokale Wert bonusPoints mit Hilfe der Datenbank initialisiert und nach dem Berechnen des neuen Punktestandes wieder rückgeschrieben.

Transaktionen	40.000
Transaktionen pro Sekunde	45.6

Tabelle 75: Überblick über die Performance-Messung Nr. 13

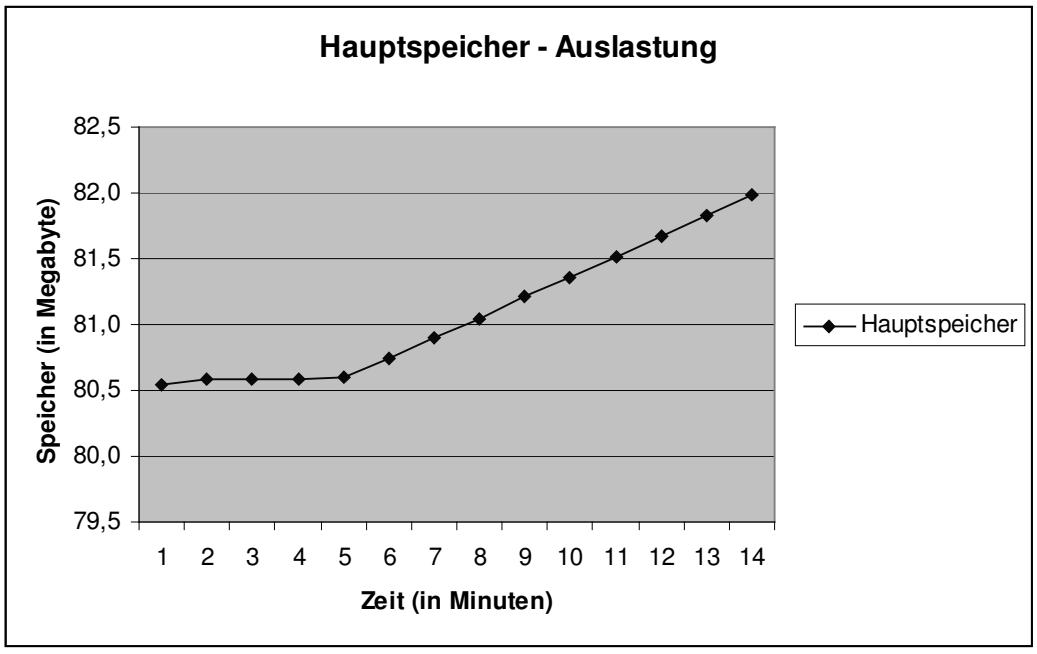

Abbildung 96: Hauptspeicher-Auslastung für Performance-Messung Nr. 13

Mittelwert	81,1

Standardabweichung	0,52
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 76: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 13

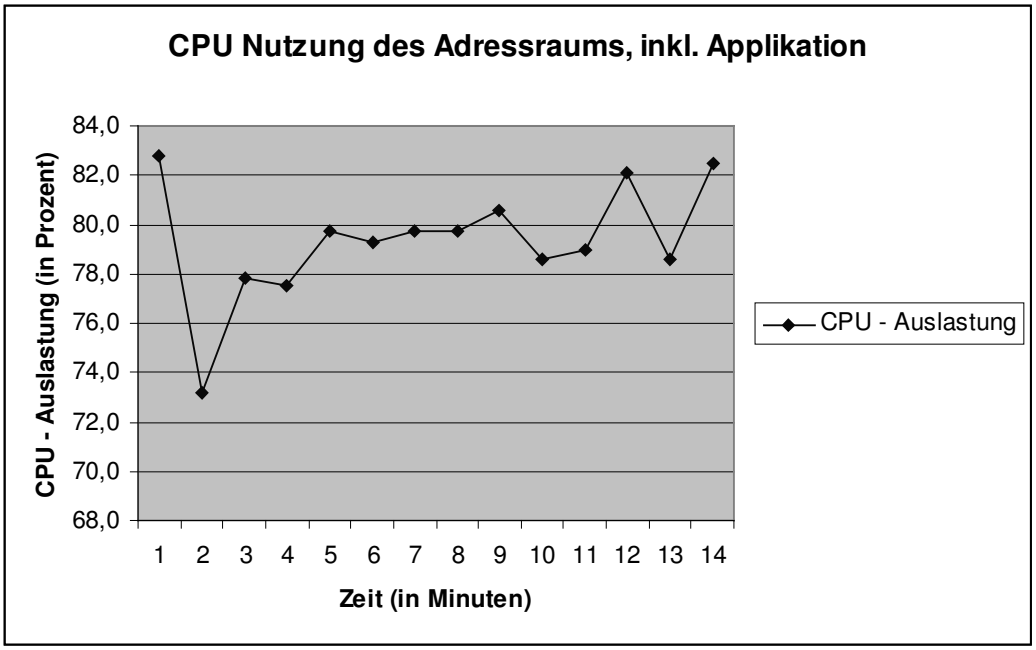

Abbildung 97: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 13

Mittelwert	79,4
Standardabweichung	2,42
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	7,1 %
Signifikanz	Marginal signifikant

Tabelle 77: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 13

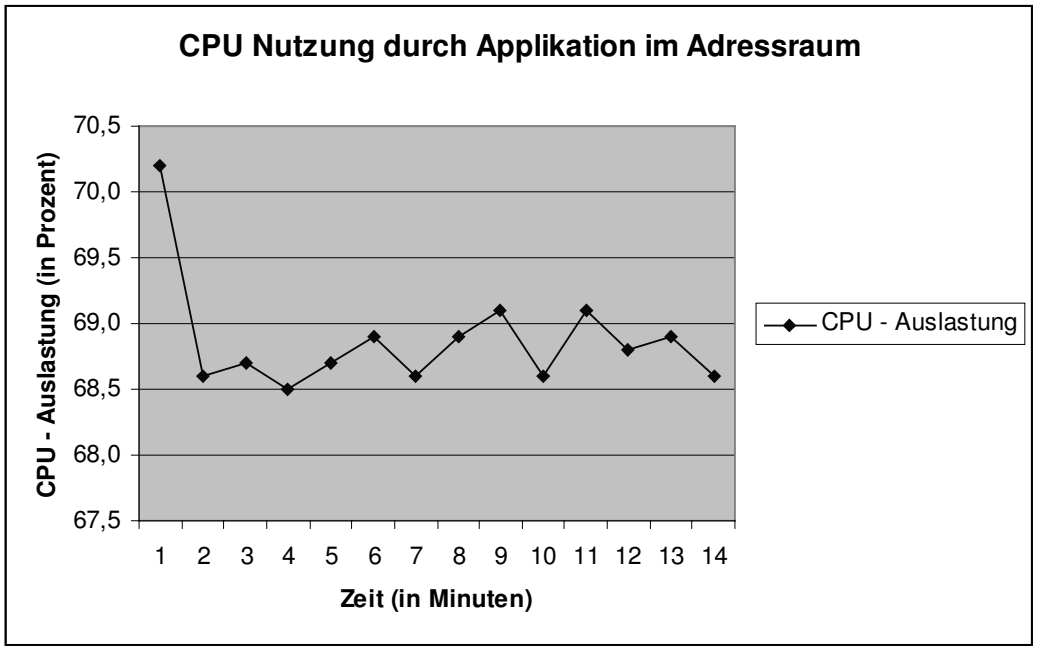

Abbildung 98: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 13

Mittelwert	68,9
Standardabweichung	0,43
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	7,1 %
Signifikanz	Marginal signifikant

Tabelle 78: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 13

amount	accountNo	shopName	rollbackTx

Abbildung 99: Globale Daten für SWF #009

balanceCustomer	balanceShop

Abbildung 100: Lokale Daten für SWF #009

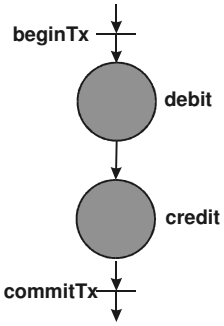

Abbildung 101: SWF #009: Zweiter Teil des SWF #007

Zu Beginn des SWF werden die lokalen Werte balanceCustomer und balanceShop mit Hilfe der Datenbank initialisiert und nach der Ausführung der beiden Überweisungen wieder rückgeschrieben.

Transaktionen	40.000
Transaktionen pro Sekunde	35,8

Tabelle 79: Überblick über die Performance-Messung Nr. 14

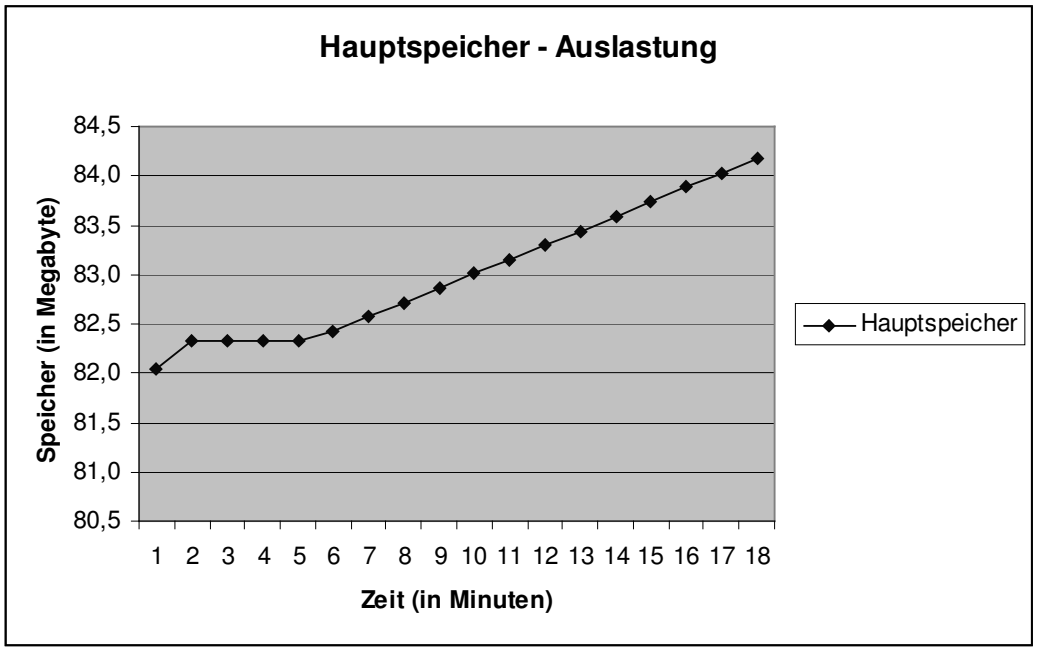

Abbildung 102: Hauptspeicher-Auslastung für Performance-Messung Nr. 14

Mittelwert	83,0
Standardabweichung	0,68
Signifikanzniveau	5,0 %

Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 80: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 14

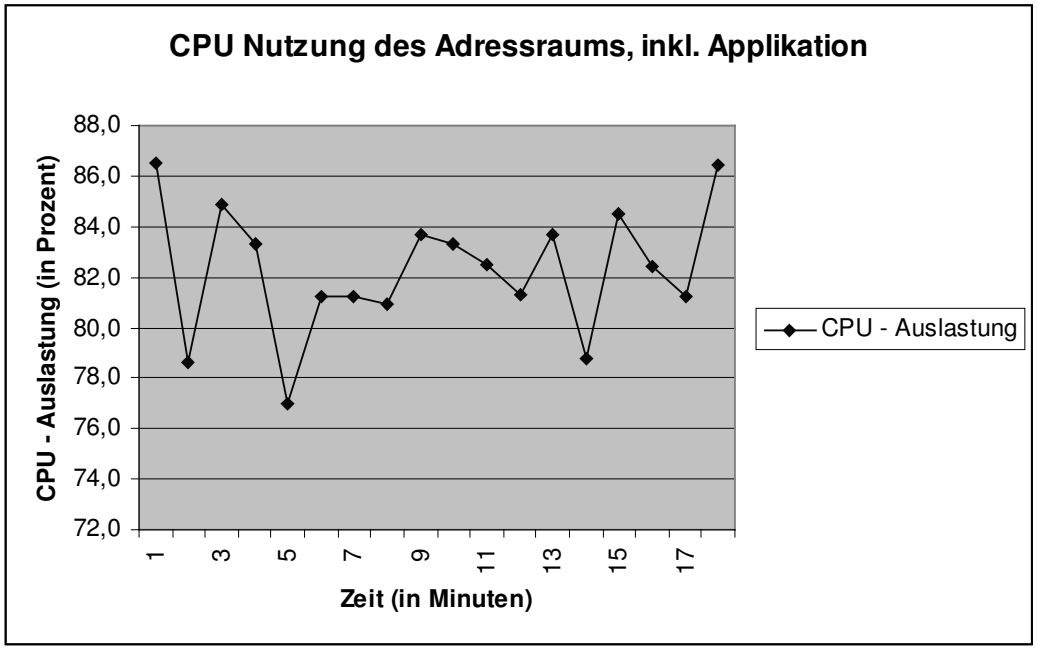

Abbildung 103: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 14

Mittelwert	82,3
Standardabweichung	2,57
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,6 %
Signifikanz	Marginal signifikant

Tabelle 81: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 14

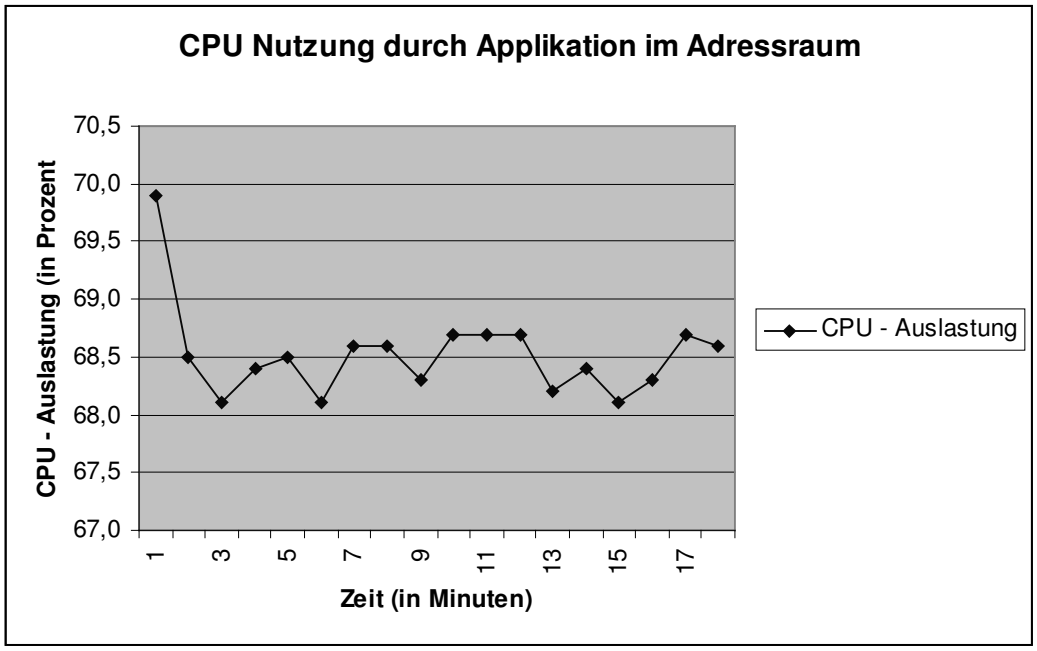

Abbildung 104: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 14

Mittelwert	68,5
Standardabweichung	0,41
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,6 %
Signifikanz	Marginal signifikant

Tabelle 82: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 14

accountNo	**shopName**

Abbildung 105: Globale Daten für SWF #010

bonusPoints	**balanceCustomer**	**balanceShop**

Abbildung 106: Lokale Daten für SWF #010

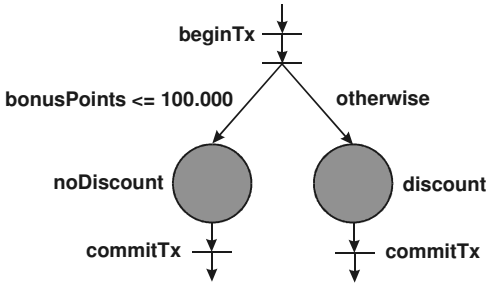

Abbildung 107: Dritter Teil des SWF #007

Bei diesem SWF werden zu Beginn die lokalen Werte bonusPoints, balanceCustomer, balanceShop und lastPurchase mit Hilfe der Datenbank initialisiert. Anschließend wird geprüft, ob der Kunde mehr als 100.000 Bonuspunkte gesammelt hat. Ist dies der Fall wird ihm 10 % der Summe von lastPurchase wieder rückerstattet. Um die Komplexität des Subworkflows zu begrenzen, werden beide Überweisungen (debit und credit) in einer Aktivität ausgeführt.

Transaktionen	40.000
Transaktionen pro Sekunde	33,9

Tabelle 83: Überblick über die Performance-Messung Nr. 15

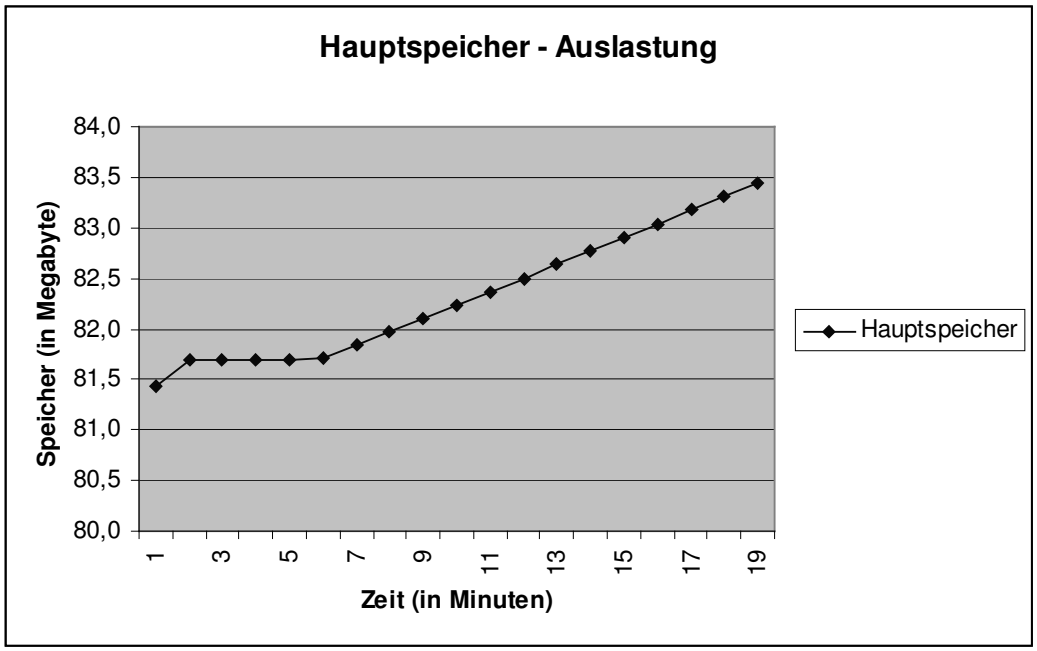

Abbildung 108: Hauptspeicher-Auslastung für Performance-Messung Nr. 15

Mittelwert	82,3
Standardabweichung	0,64

Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 84: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 15

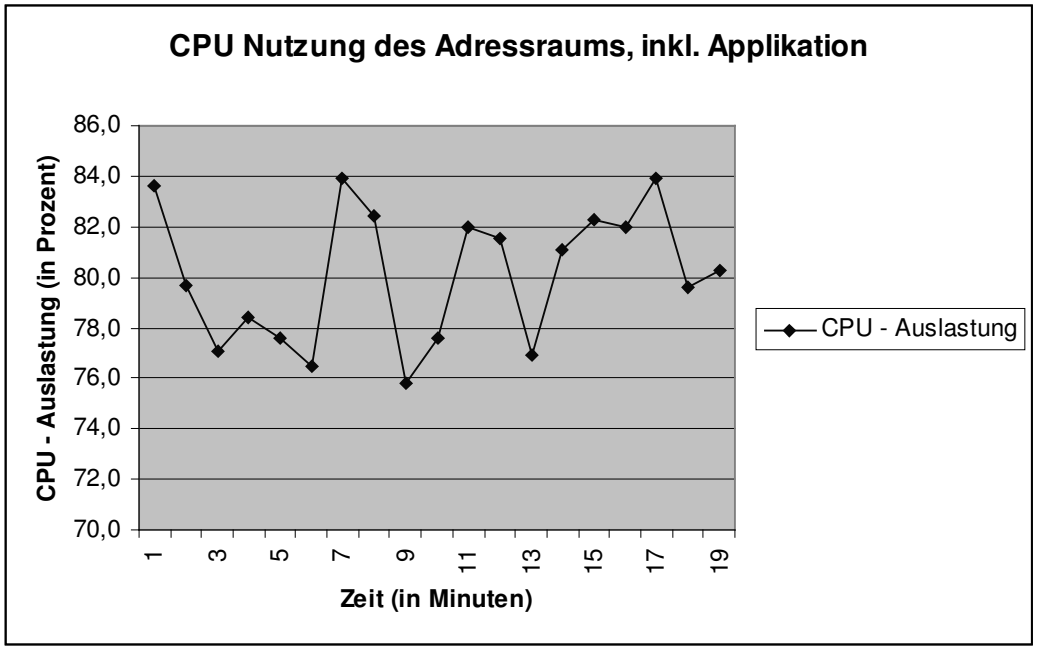

Abbildung 109: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 15

Mittelwert	80,1
Standardabweichung	2,67
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 85: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 15

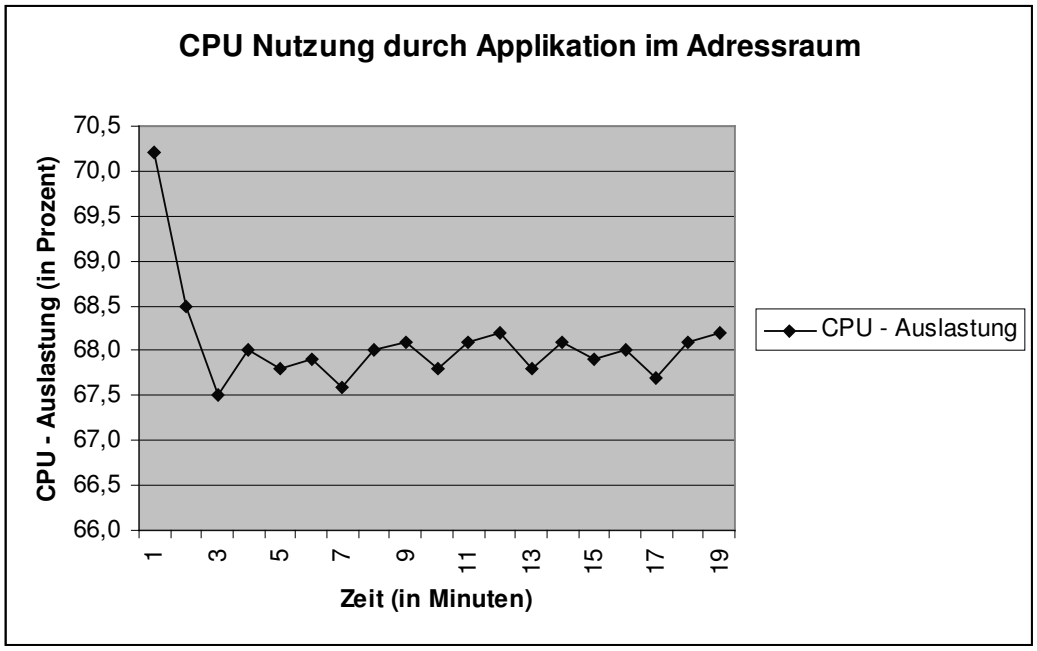

Abbildung 110: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 15

Mittelwert	68,1
Standardabweichung	0,56
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,3 %
Signifikanz	Marginal signifikant

Tabelle 86: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 15

Die Subworkflows #008-#010 entsprechen dem Ablauf des in 13.7.2.4.2 dargestellten SWF, allerdings mit dem Unterschied dass hier jede Transaktion durch einen eigenen SWF abgebildet wird.

Transaktionen	40.000
Transaktionen pro Sekunde	12,9

Tabelle 87: Überblick über die Performance-Messung Nr. 13-15, gemittelt zum Vergleich mit Performance-Test Nr. 12. Hierbei wurden die drei Subworkflows gewichtet über eine durchschnittliche Nutzung durch die Applikation im Adressraum von 68,5 %.

13.7.2.4.4 Performance-Messung Nr. 16: Einkauf in einem Internet Shop; Variante 3: SWF #008 bis SWF #010 – drei Subworkflows bestehend aus je einer Transaktion

Siehe Kapitel 13.7.2.4.3, wobei hier jeweils zuerst SWF #008, der die Bonuspunkte berecht, dann SWF #009, der die zugehörige Überweisung durchführt und zum Schluss SWF #010 direkt hintereinander ausgeführt werden.

Transaktionen	40.000
Transaktionen pro Sekunde	12,4

Tabelle 88: Überblick über die Performance-Messung Nr. 16

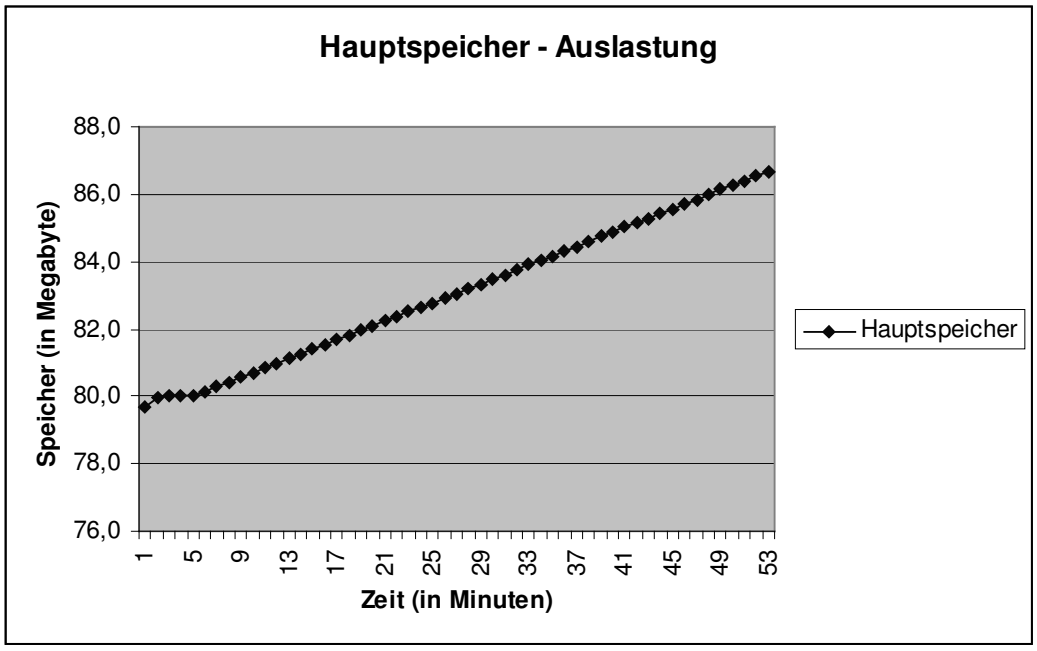

Abbildung 111: Hauptspeicher-Auslastung für Performance-Messung Nr. 16

Mittelwert	83,1
Standardabweichung	2,12
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 89: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 16

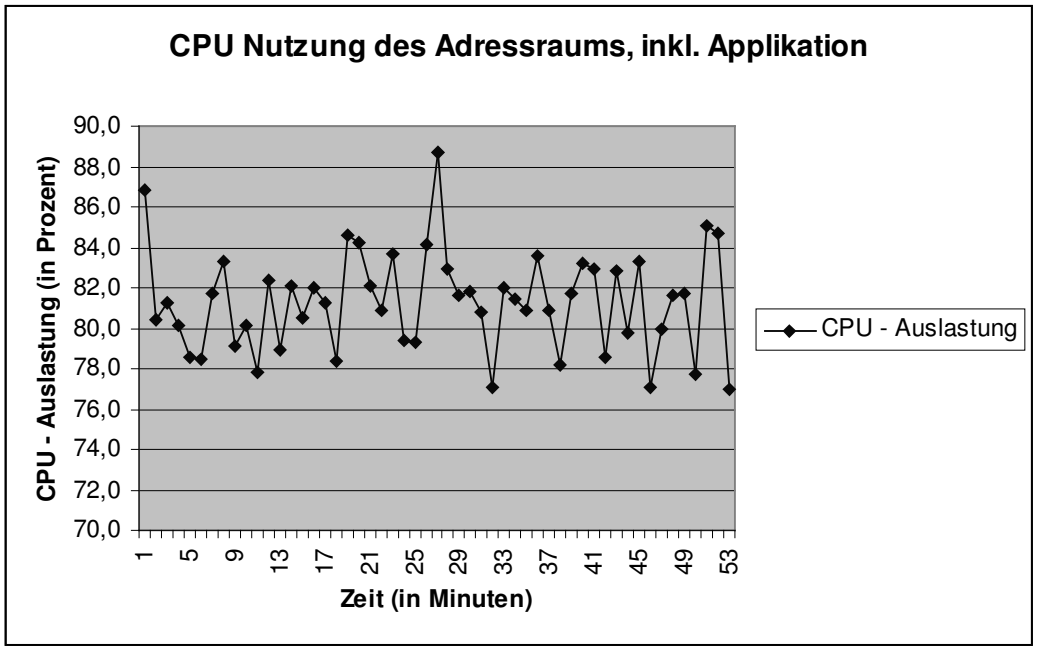

Abbildung 112: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 16

Mittelwert	81,3
Standardabweichung	2,48
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	3,8 %
Signifikanz	Signifikant

Tabelle 90: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 16

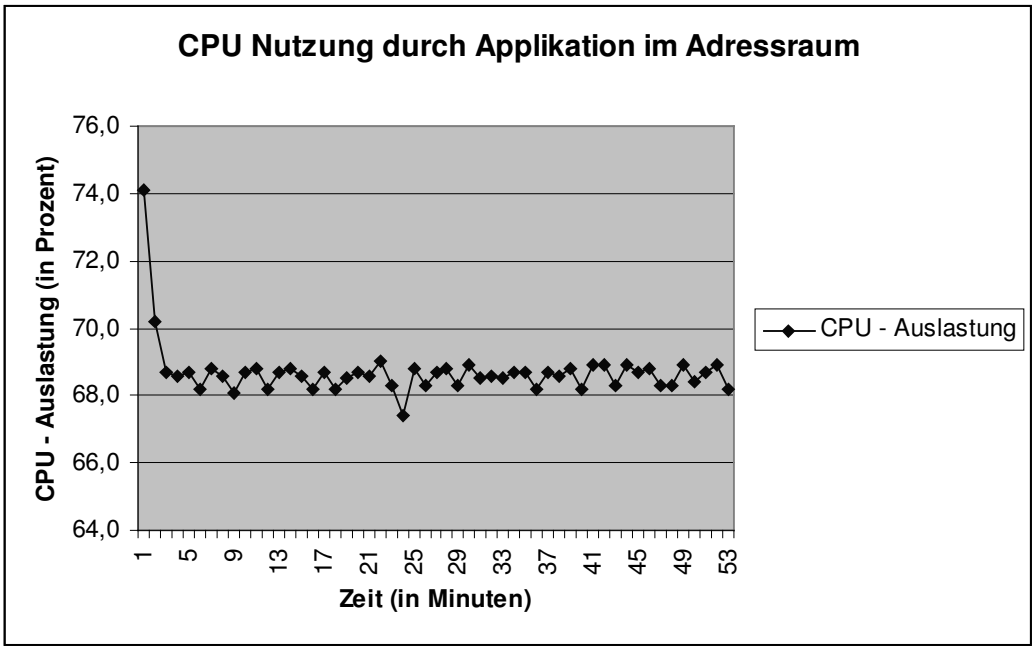

Abbildung 113: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 16

Mittelwert	68,7
Standardabweichung	0,84
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	1,9 %
Signifikanz	Signifikant

Tabelle 91: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 16

13.7.2.4.5Performance-Messung Nr. 17: Einkauf in einem Internet Shop; Variante 4: SWF #011 – ein SWF bestehend aus einer Transaktion

amount	accountNo	shopName	rollbackTx

Abbildung 114: Globale Daten für SWF #011

bonusPoints	balanceCustomer	balanceShop

Abbildung 115: Lokale Daten für SWF #011

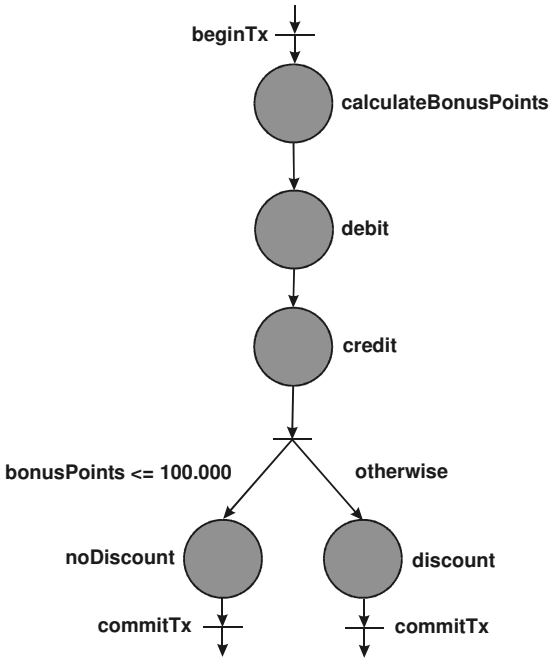

Abbildung 116: SWF #011: SWF #007 bestehend aus einer Transaktion

Der SWF #011 entspricht dem in dargestellten SWF, allerdings mit dem Unterschied, dass dieser SWF nicht aus drei sondern aus einer Transaktion besteht.

Transaktionen	40.000
Transaktionen pro Sekunde	28,7

Tabelle 92: Überblick über die Performance-Messung Nr. 17

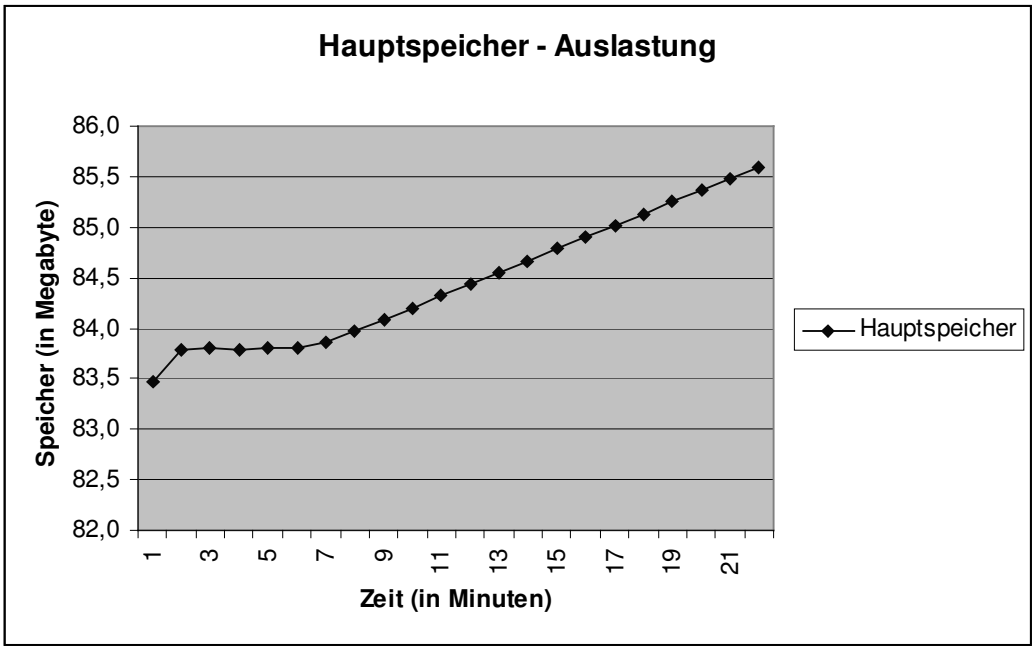

Abbildung 117: Hauptspeicher-Auslastung für Performance-Messung Nr. 17

Mittelwert	84,5
Standardabweichung	0,65
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 93: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 17

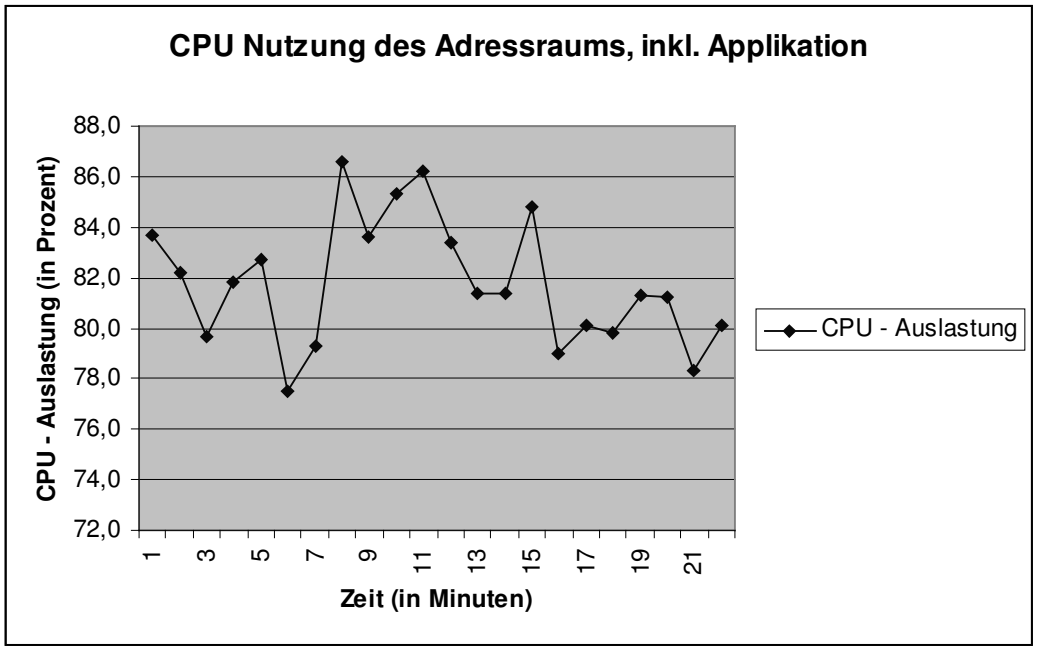

Abbildung 118: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 17

Mittelwert	81,8
Standardabweichung	2,53
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 94: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 17

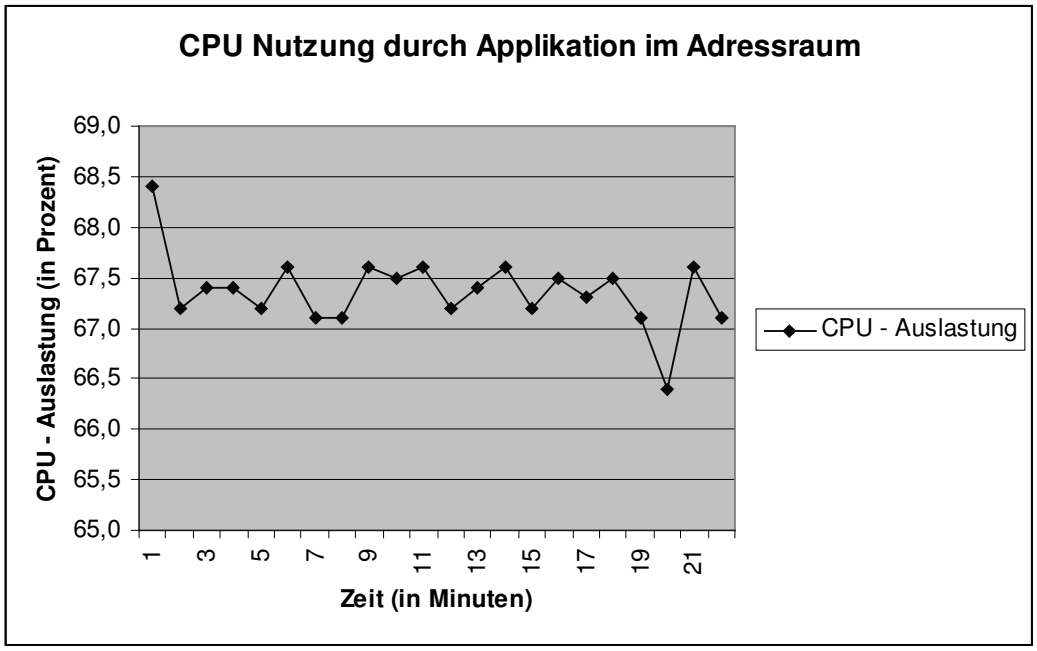

Abbildung 119: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 17

Mittelwert	67,4
Standardabweichung	0,36
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	9,1 %
Signifikanz	Marginal signifikant

Tabelle 95: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 17

13.7.2.4.6Abbruch Szenarien

13.7.2.4.6.1 Performance-Messung Nr. 18: Einkauf in einem Internet Shop; Variante 5: SWF #007 – 25 % der Aufrufe der credit-Aktivität enden mit dem Return Code RC_ROLLBACKTX

In 25 % der Fälle ist der globale Wert rollbackTx „true“, was dazu führt, dass 25 % der credit-Aufrufe mit dem Return Code RC_ROLLBACKTX enden. Ansonsten verhält sich der SWF analog wie in 13.7.2.4.1 und 13.7.2.4.2 beschrieben.

Transaktionen	40.000
Transaktionen pro Sekunde	16,5

Tabelle 96: Überblick über die Performance-Messung Nr. 18

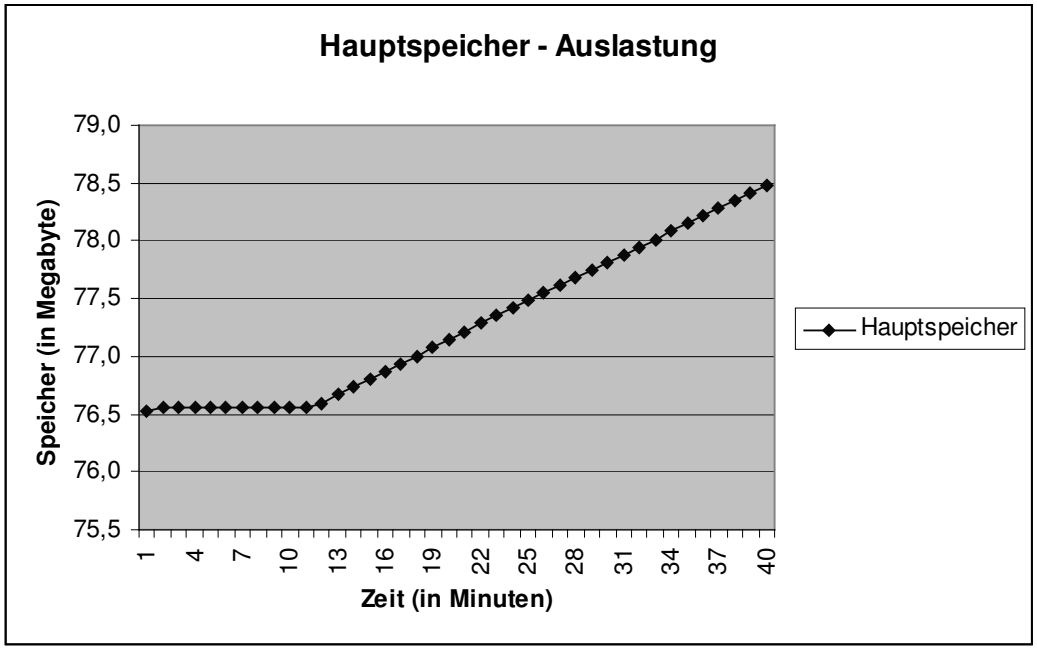

Abbildung 120: Hauptspeicher-Auslastung für Performance-Messung Nr. 18

Mittelwert	77,3
Standardabweichung	0,66
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 97: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 18

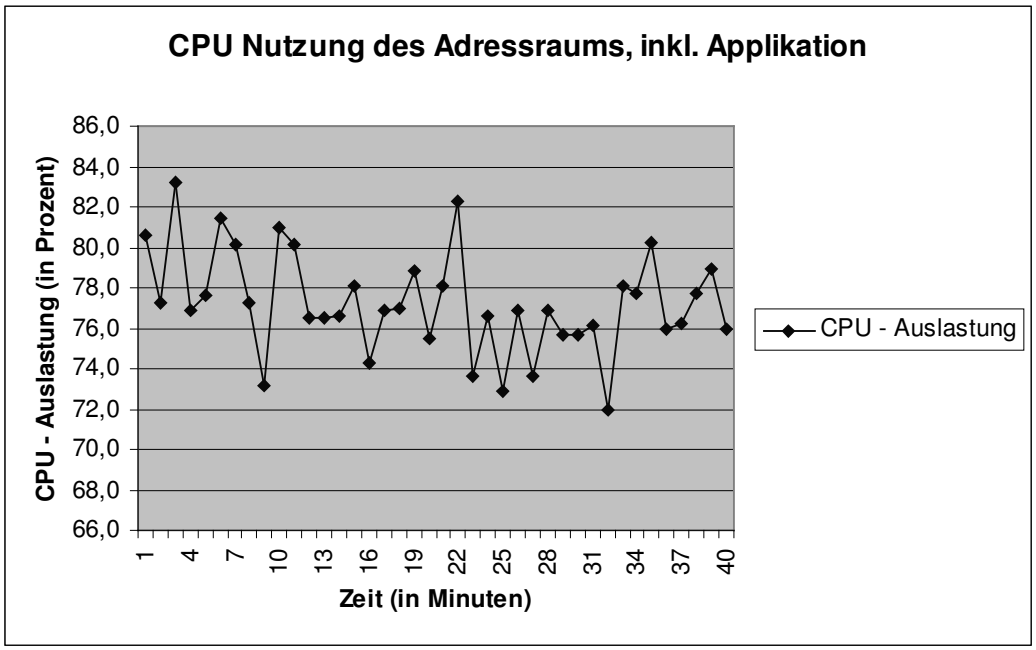

Abbildung 121: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 18

Mittelwert	77,3
Standardabweichung	2,53
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	5,0 %
Signifikanz	Signifikant

Tabelle 98: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 18

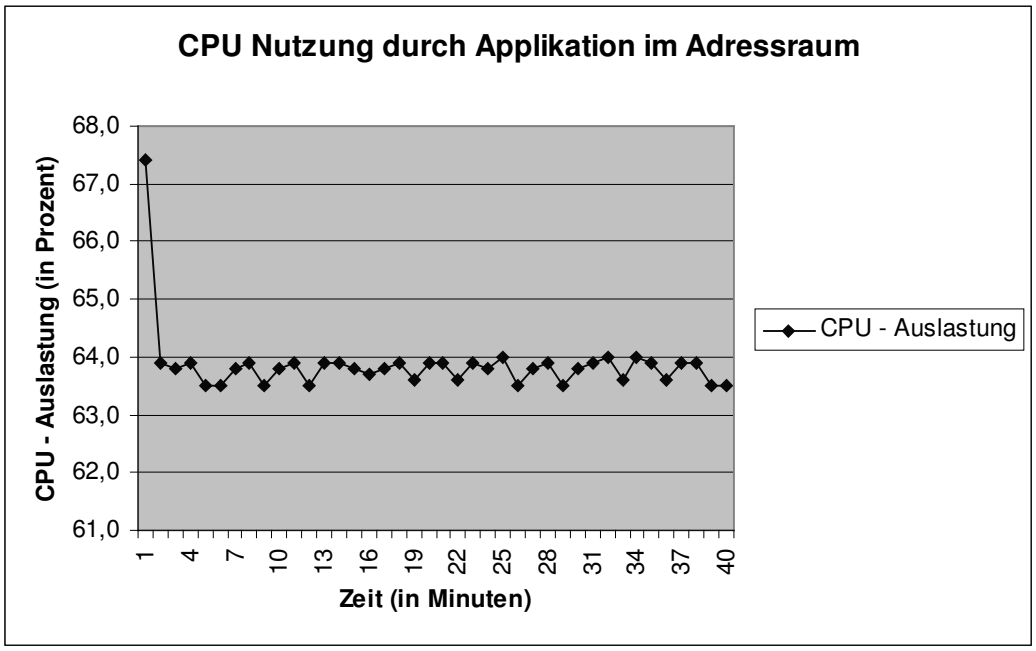

Abbildung 122: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 18

Mittelwert	63,9
Standardabweichung	0,60
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	2,5 %
Signifikanz	Signifikant

Tabelle 99: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 18

13.7.2.4.6.2 Performance-Messung Nr. 19: Einkauf in einem Internet Shop; Variante 6: SWF #007 – 50 % der Aufrufe der credit-Aktivität enden mit dem Return Code RC_ROLLBACKTX

In 50 % der Fälle ist der globale Wert rollbackTx „true“, was dazu führt, dass 50 % der credit-Aufrufe mit dem Return Code RC_ROLLBACKTX enden. Ansonsten verhält sich der SWF analog wie in 13.7.2.4.1 und 13.7.2.4.2 beschrieben.

Transaktionen	40.000
Transaktionen pro Sekunde	16,5

Tabelle 100: Überblick über die Performance-Messung Nr. 19

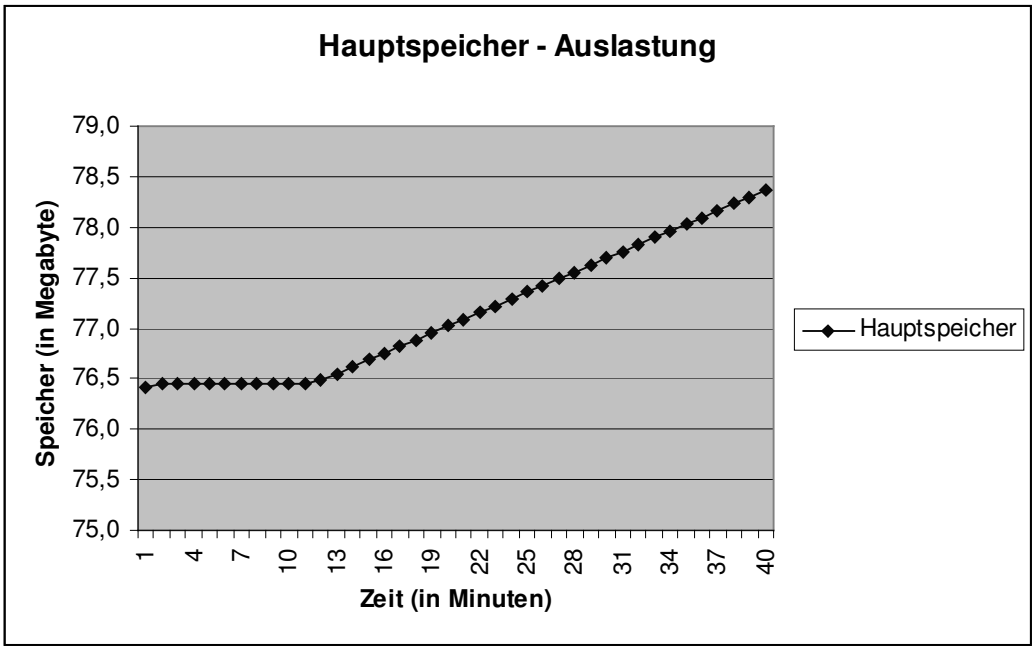

Abbildung 123: Hauptspeicher-Auslastung für Performance-Messung Nr. 19

Mittelwert	77,2
Standardabweichung	0,65
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	0,0 %
Signifikanz	Sehr signifikant

Tabelle 101: Statistische Kennzahlen für die Hauptspeicher-Auslastung bei Performance-Messung Nr. 19

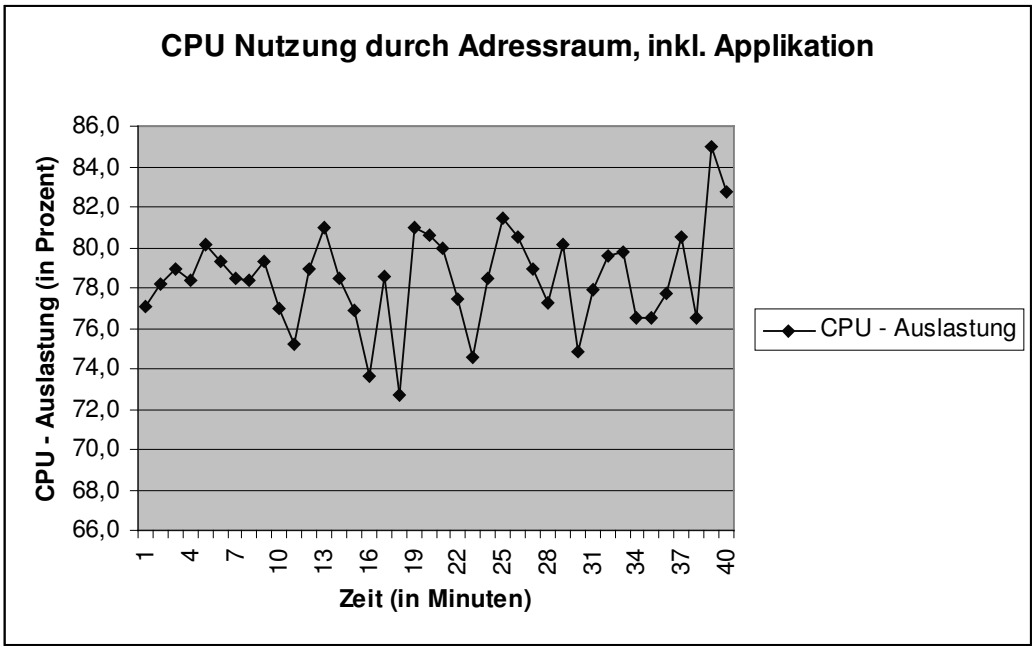

Abbildung 124: CPU Nutzung des Adressraums, inkl. Applikation für Performance-Messung Nr. 19

Mittelwert	78,5
Standardabweichung	2,40
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	7,5 %
Signifikanz	Marginal signifikant

Tabelle 102: Statistische Kennzahlen für die CPU Nutzung des Adressraums (inkl. Applikation) bei Performance-Messung Nr. 19

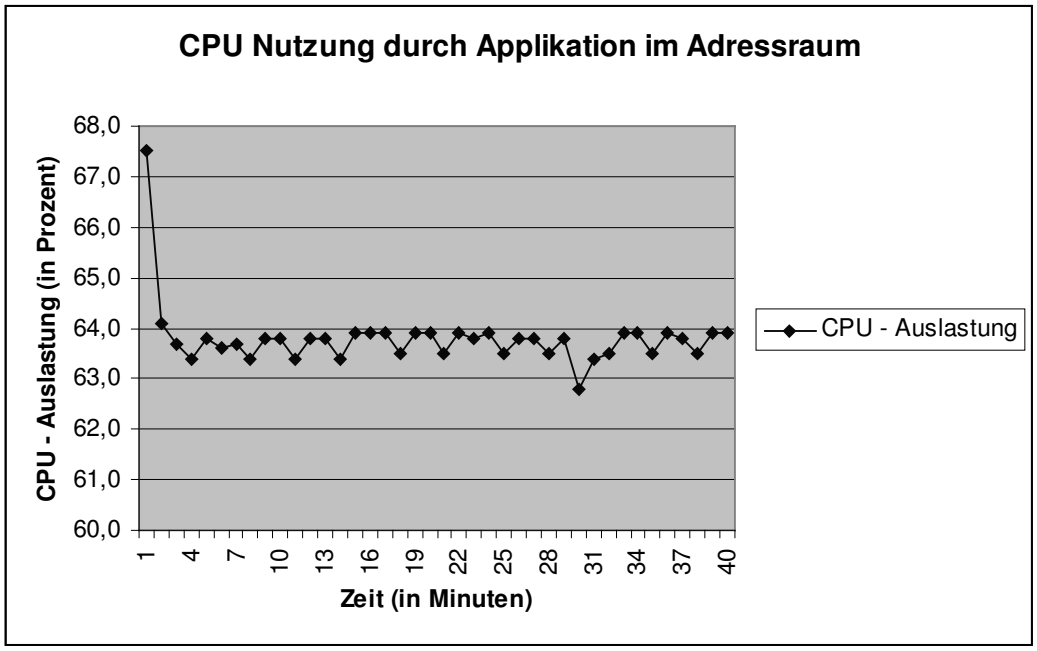

Abbildung 125: CPU Nutzung durch Applikation im Adressraum für Performance-Messung Nr. 19

Mittelwert	63,8
Standardabweichung	0,65
Signifikanzniveau	5,0 %
Irrtumswahrscheinlichkeit	2,5 %
Signifikanz	Signifikant

Tabelle 103: Statistische Kennzahlen für die CPU Nutzung durch die Applikation im Adressraum bei Performance-Messung Nr. 19

13.8 Beispiel-Konfigurationsdatei für die Subworkflowsteuerung

```
# application package name
"com.thornung.swfc.application"
# caching of object and method references flag
false
# number of retries for COBOL activites; return code = ERRORxxx
3
# number of retries for COBOL activites; return code = RECOVER
3
```

```
# number of retries for COBOL activites; return code = RESTORETX
3
# number of retries for COBOL activites; return code = ROLLBACKTX
3
# cobol library path
"/u/hornung/SWFC/COBOL/"
# number of retries for COBOL plugins; return code = RECOVER
3
# database data filename
"database.swfc"
# data description base name
"DataDescription"
# data section copystruktur base name
"DataSectionCopystruktur"
# number of runs after which garbage collection should take place
128
# number of retries for Java activites; return code = ERRORxxx
3
# number of retries for Java activites; return code = RECOVER
3
# number of retries for Java activites; return code = RESTORETX
3
# number of retries for Java activites; return code = ROLLBACKTX
3
# number of retries for Java plugins; return code = RECOVER
3
# local savepoint filename
"localSavepoint.swfc"
# local view class name
"LocalView"
# middleware package name
"com.thornung.swfc.middleware"
# Performance-Test flag for Container Factory
false
```

```
# Performance-Test flag for Data Service
false
# Performance-Test flag for I/O Controller
false
# Performance-Test flag for JNI/COBOL Facility
false
# Performance-Test flag for SWFC
false
# Performance-Test flag for SWFDLParser
false
# Performance-Test flag for SWF Interpreter
false
# base uri for savefiles
"/u/hornung/SWFC/savefiles/"
# SAX Parser Class Name
"org.apache.xerces.parsers.SAXParser"
# base uri for swfdl files
"/u/hornung/SWFC/bpel/"
# extension for swfdl files (default = .xml)
".xml"
# SWFC RWU savepoint filename
"swfcRWUSavepoint.swfc"
# Test Driver -> mode
1
# Test Driver -> number of transactions
20000
# Test Driver -> which Test Driver
# 0 == Marc Beyerle´s Test Driver
# 1 == Java Test Driver
0
# transactional savepoint filename
"transactionalSavepoint.swfc"
# verbose flag for Container Factory
false
```

```
# verbose flag for Data Service
false
# verbose flag for I/O Controller
false
# verbose flag for JNI/COBOL Facility
false
# verbose flag for SWFC
false
# verbose flag for SWFDL Parser
false
# verbose flag for SWF Interpreter
false
```

13.9 Konfigurationsdatei der PRJVM, die bei den Performance-Messungen verwendet wurde

```
##########################################################################
#
# Instructions for modifying this file for your own use:
#
# 1. Replace all occurrences of /opt/IBMJava2-131 with the path to
# your JDK installation, if different.
# 2. Replace all occurrences of
# home/marc/Uni/Diplomarbeit/OnlineBanking with the path to your
# Java classes.
#
##########################################################################
#
# Environment Settings
#
# This section is where we declare environment variables for ALL
# subsequent JVMs that we create (Master AND Worker). env.var.count is
# required if any env.var properties are to be used.
#
##########################################################################
```

```
env.var.count=6
env.var.1=_BPX_SHAREAS=MUST
env.var.2=JAVA_HOME=/SYSTEM/local/java/J1.3/IBM/J1.3
env.var.3=LIBPATH=/SYSTEM/local/java/J1.3/IBM/J1.3/bin:/SYSTEM/local/j
ava/J1.3/IBM/J1.3/bin/classic:/SYSTEM/local/db2/db2v7/db2/db2710/lib:$
LIBPATH
env.var.4=LD_LIBRARY_PATH=/SYSTEM/local/java/J1.3/IBM/J1.3/bin:/SYSTEM
/local/java/J1.3/IBM/J1.3/bin/classic:/SYSTEM/local/db2/db2v7/db2/db27
10/lib:$LD_LIBRARY_PATH
env.var.5=DB2SQLJPROPERTIES=/u/hornung/db2sqljjdbc.properties
env.var.6=DSNAOINI=/u/hornung/hornung.dsnaoini
#########################################################################
#
# Monitor Settings
#
# This section is used for miscellaneous properties. The following
# is a list of all possible properties for this section.
#
# log.name=<filename>
# workers.count=<value>
# communicator.port=<value>
#
#########################################################################
log.name=jvmset.log
workers.count=1
communicator.port=54321
#########################################################################
#
# Master Settings
#
# This section is used to set various properties for the Master JVM
# if running a JVMSet. The following is a list of all properties
# available to be set. -Xjvmset[token] must be specified here for this
# to work!
#
```

```
# master.options={-Xargs}
# master.options.properties={-Dargs}
#
#########################################################################
master.options=-Xms64M -Xmx512M -Xresettable -Xjvmset64M
master.options.properties= -
Dibm.jvm.trusted.middleware.class.path=/SYSTEM/local/db2/db2v7/db2/db2
710/classes:/SYSTEM/local/db2/db2v7/db2/db2710/classes/db2sqljruntime.
zip:/u/hornung/SWFC/middleware:/u/hornung/SWFC/middleware/xercesImpl.j
ar:/u/hornung/SWFC/middleware/xercesSamples.jar:/u/hornung/SWFC/middle
ware/xmlParserAPIs.jar -
Dibm.jvm.shareable.application.class.path=/u/hornung/SWFC/application
#########################################################################
#
# Workers' Settings
#
# This section is used to set various properties for Worker JVMs.
# Worker properties are set in the following way:
# worker.n.property=<value> where 0 < n <= workers.count
#
# It is important to have properties set up for the Worker JVMs.
# -Xjvmset must be specified here for this to work! Of course,
# -Xresettable must be set for
# worker.n.reset.interval=<value> to be meaningful.
#
# worker.n.options={-Xargs}
# worker.n.options.properties={-Dargs}
# worker.n.reset.interval=<value>
#
#########################################################################
#
# Worker JVM #1
#
#########################################################################
worker.1.options=-Xresettable -Xjvmset -Xms10M -Xmx192M
worker.1.options.properties=
worker.1.reset.interval=1
```

##