Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D – 70569 Stuttgart

Fachstudie Nr. 106

Vergleich von Ansätzen zur Fragmentierung von Prozessen

Tobias Binz, Jan Königsberger, Tina Schliemann

Studiengang:	Softwaretechnik
Prüfer:	Dr.-Ing. Dimka Karastoyanova, Juniorprofessor
Betreuer:	Dipl.-Inf. Olga Danylevych
begonnen am:	29.06.2009
beendet am:	29.09.2009

CR-Klassifikation: H.4.1 Workflow Management

Inhaltsverzeichnis

1 Einleitung ... 3
1.1 Anwendungsfelder der Fragmentierung von Prozessen ... 3
1.2 Aufbau des Dokuments ... 4
2 Ansätze und Verfahren ... 5
2.1 Einteilung der Papers ... 5
2.2 Überblick über verschiedene Fragmentierungs-Ansätze ... 6
2.3 Ausgewählte Ansätze ... 23
3 Kriterienkatalog ... 24
3.1 Grundlagen ... 24
3.2 Wie wird im Ansatz fragmentiert? ... 24
3.3 Anwendbarkeit des Ansatzes ... 24
3.4 Ausführbarkeit des Ansatzes (falls zutreffend) ... 25
4 Bewertung und Vergleich der Ansätze ... 26
4.1 Anhand des Kriterienkatalogs ... 26
4.2 Tabellarischer Vergleich ausgewählter Gesichtspunkte ... 31
5 Szenario: Anwendung der Ansätze auf ein Prozessmodell ... 32
5.1 Das Szenario: "LoanApprovalExtended" ... 32
5.2 A: Role-based Decomposition of Business Processes using BPEL ... 33
5.3 B: A Novel Approach to Decentralized Workflow Enactment ... 35
5.4 C: Optimal Stratification of Transactions ... 38
5.5 I: Decentralizing execution of composite Webservices ... 40
6 Fazit ... 45
7 Abbildungsverzeichnis ... 46
8 Tabellenverzeichnis ... 46
9 Literaturverzeichnis ... 47
10 Anhang ... 49
10.1 LoanApprovalExtended.bpel ... 49
10.2 LoanApprovalExtended.wsdl ... 52

Abbildung 1: Tagcloud der am häufigsten verwendeten Wörter (wordle.net)

1 Einleitung

Die Fragmentierung von Geschäftsprozessen wird in Unternehmen immer wichtiger, da sie eine schnelle Anpassung und Neuentwicklung von Prozessen unterstützt, welche heutzutage entscheidend für den Markterfolg eines Unternehmens sind ("Prozess = Produkt"). In dieser Fachstudie werden deshalb verschiedene Ansätze zur Fragmentierung von Geschäftsprozessen untersucht, bewertet und verglichen.

Das Fragment eines Geschäftsprozesses ist eine zusammenhängende Untermenge von Aktivitäten dieses Prozesses. Wie genau diese Menge bestimmt wird und ob das Fragment für sich ausführbar ist oder nicht, ist von Ansatz zu Ansatz unterschiedlich. In den nun folgenden Anwendungsfeldern wird sich zeigen, dass bereits dieses einfache Modell viele interessante Möglichkeiten eröffnet.

1.1 Anwendungsfelder der Fragmentierung von Prozessen

Fragmentierung kann sehr vielfältig eingesetzt werden. Hier wollen wir eine, sicher nicht vollständige, Liste einiger Anwendungsfelder aufzeigen, die auch teilweise in den untersuchten Ansätzen genannt wurden:

- Bei der Prozessoptimierung in Unternehmen können einzelne Fragmente, die nicht mehr wirtschaftlich im Unternehmen ausgeführt werden können, ausgelagert werden.
  Auch bei Kooperationen verschiedener Unternehmen, die ihre Abläufe durch firmenübergreifende Geschäftsprozesse optimieren wollen, ist es nötig, dass Fragmente, welche interne Daten eines Unternehmens benötigen, auch ausschließlich auf der IT Infrastruktur dieses Unternehmens ausgeführt werden.

- Um einen höheren Durchsatz, kürzere Antwortzeiten, bessere Skalierbarkeit etc. zu erreichen, können einzelne Fragmente eines Geschäftsprozesses verteilt ausgeführt werden.

- Komplexe Prozesse sind kaum noch beherrschbar und das Wissen zur Modellierung ist oft nicht bei einer Person vorhanden (sondern beispielsweise in verschiedenen Fachabteilungen verteilt). Durch Fragmentierung kann der Prozess in mehrere kleinere Fragmente aufgeteilt werden, deren Komplexität dann überschaubarer ist. Diese Fragmente lassen sich dezentralisiert modellieren ("divide & conquer"), was Fehler vermeidet und Zeit spart.

- Teile eines Prozesses können als Fragment extrahiert und zur Wiederverwendung in einen Baukasten (Workflow-Patterns, Complience-Vorgaben etc.) aufgenommen werden.

- "Mobile Worker" können Prozessfragmente auf ihrem Notebook auschecken, offline verwenden und später die Änderungen wieder einchecken.

- Während der Modellierung kann Fragmentierung zur schnelleren Prüfung der Korrektheit (Kontrollflussanalyse) oder zum Finden von passenden Teilstücken (Auto-Ergänzung) verwendet werden.

1.2 Aufbau des Dokuments

In Kapitel 2 beschäftigen wir uns mit verschiedenen Ansätzen und teilen diese in Kategorien ein. Zu jedem der Papers befindet sich dort außerdem eine Zusammenfassung, die beschreibt, wie die Fragmentierung im jeweiligen Ansatz funktioniert und die Besonderheiten, Nachteile und Vorteile des jeweiligen Ansatzes beleuchtet. Einige dieser Papers werden anschließend genauer untersucht und miteinander verglichen.

In Kapitel 3 werden die Kriterien beschrieben, nach welchen die Ansätze bewertet werden.

Kapitel 4 schließt daran mit einer Bewertung der Papers (anhand der Kriterien aus Kapitel 3) an.
Über die Kriterien hinausgehend werden einige wichtige Sachverhalte in Übersichtstabellen dargestellt, die nicht als Vergleichskriterien dienen können. Hierzu zählt unter anderem, in welchem Format das Prozessmodell vorliegen muss, um einen Ansatz darauf anwenden zu können.

In Kapitel 5 wird ein Szenario vorgestellt, an dem die gewählten Fragmentierungsansätze durchgeführt werden. Die Durchführung und deren Ergebnisse haben wir für jeden Ansatz skizziert.

In Kapitel 6 wird mit einem Fazit und der Bewertung dieser Arbeit geschlossen.

Abschließend folgen noch Abbildungs- (Kapitel 7), Tabellen- (Kapitel 0) und Literaturverzeichnis (Kapitel 9). Im Anhang (Kapitel 10) ist des Weiteren der Quell-Code des Beispiel-Szenarios zu finden.

2 Ansätze und Verfahren

In dieser Fachstudie wurden eine Reihe von Papers, die sich mit verschiedensten Ansätzen zur Fragmentierung von Prozessen beschäftigen, gesammelt und jeweils kurz zusammengefasst, um einen allgemeinen Überblick zu geben. Aus diesen Ansätzen wurden vier besonders interessante für eine genauere Untersuchung ausgewählt, mithilfe eines Kriterienkatalogs bewertet und verglichen.

Folgende 12 Papers haben wir in unserer Fachstudie betrachtet, zur leichteren Identifizierung, sind die Paper alphabetisch nummeriert.

A. Rania Khalaf, Frank Leymann
Role-based Decomposition of Business Processes using BPEL [1]

B. Daniel Martin, Daniel Wutke, Frank Leymann
A Novel Approach to Decentralized Workflow Enactment [2]

C. Olha Danylevych, Dimka Karastoyanova, Frank Leymann
Optimal Stratification of Transactions [3]

D. Zhilei Ma, Frank Leymann
BPEL Fragments for Modularized Reuse in Modeling BPEL Processes [4]

E. Artem Polyvyanyy, Sergey Smirnov, Mathias Weske
On Application of Structural Decomposition for Process Model Abstraction [5]

F. Kwang-Hoon Kim, Jae-Kang Won, Chang-Min Kim
A Fragment-Driven Process Modeling Methology [6]

G. Frank Lindert, Wolfgang Deiters
Modelling inter-organizational processes with process model fragments [7]

H. Fabio Casati, Angela Discenza
Supporting Workflow Cooperation Within and Across Organizations [8]

I. Mangala Gowri Nanda, Satish Chandra, Vivek Sarkar
Decentralizing execution of composite Webservices [9]

J. Volker Guth, Kirsten Lenz, Andreas Oberweis
Distributed Workflow Execution based on fragmentation of Petri Nets [10]

K. Wei Tan, Yushun Fan
Dynamic workflow model fragmentation for distributed execution [11]

L. Jussi Vanhatalo, Hagen Völzer, Frank Leymann
Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition [12]

2.1 Einteilung der Papers

Die Ziele welche die Papers verfolgen, können in diese Themenbereiche eingeteilt werden:

- Verteile Ausführung zur Verbesserung der Quality of Services
  - A: Prozessmodellierer kann Aktivitäten anhand der Business-Anforderung auf die Fragmente verteilen. Der verteilt ausführbare BPEL-Code wird dann generiert.
  - B: Ausführung als Petri-Netz mit zentralem Datenspeicher (Tuplespace)
  - C: Aufteilung des Prozessmodells in transaktionsbasierte Schichten

    - I:Verteilung des BPEL-Codes auf die Server mit den Webservices
    - J: Fragmentierung mit Hilfe von Petri-Netzen zur Design-Zeit
    - K: Fragmentierung mit Hilfe von Petri-Netzen zur Laufzeit
- Analyse mit Hilfe von *Single-Entry-Single-Exit*-Blöcken (SESE)
    - E: Abstraktion von Prozessmodellen in unterschiedlichen Detail-Graden zur Analyse unterschiedlichster Aspekte
    - L: Fragmentierung in SESE-Blöcke zur effizienten Analyse der einzelnen Fragmente
- Wiederverwendung
    - D: Erstellung von wiederverwendbaren Fragmenten, erweitert die Möglichkeiten von Abstract BPEL
- Verteilte Modellierung
    - F: Firmenübergreifende Fragmentierung, die die Geheimhaltung der firmeninternen privaten Prozesse gewährleistet.
    - G: Jede Organisationseinheit ist für ihr eigenes Fragment verantwortlich. Alle Fragmente können zusammen als ein Prozess ausgeführt werden.

Leider enthält Paper [H] nicht, wie zuerst gedacht, einen Fragmentierungsansatz und wird deshalb, bis auf die Zusammenfassung, in der weiteren Fachstudie nicht betrachtet.

2.2 Überblick über verschiedene Fragmentierungs-Ansätze

2.2.1 A: Role-based Decomposition of Business Processes using BPEL

Ziel dieses Verfahren ist es, vorhandene BPEL Workflows so zu fragmentieren, dass die Fragmente von verschiedenen Teilnehmern ausgeführt werden können. Diese Fragmente können auch ohne Probleme an Dritte ausgelagert werden ("Outsourcing"). Der Prozessmodellierer kann, ohne Wissen der technischen Hintergründe und technischen Zwänge, die einzelnen Aktivitäten auf „Swimlanes“ (eine pro Teilnehmer) verteilen. Der vorgestellt Ansatz kümmert sich um die technische Umsetzung dieser Partitionierung, indem Daten- und Kontrollflüsse in Nachrichtenflüsse transformiert werden.

Eingabe und Ausgabe der Transformation

Die Transformation benötigt als Eingabe die WSDL-Dateien (welche in BPEL referenziert werden), ein Mapping (welcher Teilnehmer welche Aktivitäten ausführt) und ein BPEL-Prozessmodell, welches aber zu Beginn in BPEL-D umgewandelt wird. BPEL-D ist eine Untermenge von BPEL, deren Einschränkung hauptsächlich darin besteht, „gescopete“ Variablen (BPEL Variables) durch explizite Datenlinks und Datenfluss zu ersetzt. Deshalb gelten folgende Einschränkungen: „Dead Path Elimination“ (DPE) für den Prozess muss aktiviert sein, es gibt genau ein Correlation Set, es gibt nur eine Flow-Activity auf oberster Ebene, fast alle Simple-Activities von BPEL sind erlaubt (außer *terminate*, *throw*, *compensate* und ein *assign* welches in einen Endpunkt kopiert) und ein Request-Reply-Paar darf nicht in unterschiedlichen Fragmenten platziert sein. Das verhindert unter anderem dynamisches Binden von Partnern, d.h. alle Partner müssen zum Deployment gebunden werden.
Das Ergebnis der Transformation sind ein BPEL-Prozessmodell, eine WSDL-Datei pro Teilnehmer (enthält PartnerLinks zu allen Teilnehmern, die PartnerLinks der Original-WSDL, welche in dem Fragment benötigt werden, die benötigten Variablen und ein CorrelationSet) und Informationen zum Deployment der neuen Prozesse.

Durchführung der Fragmentierung

Bei der eigentlichen Fragmentierung wird jedem Teilnehmer mindestens eine Aktivität zur Bearbeitung zugeteilt, wobei jede Aktivität nur einem Teilnehmer zugewiesen sein darf. Die wichtigste Frage ist nun, wie die aufgebrochenen Links unter Verwendung der bekannten BPEL Konstrukte wieder verbunden werden können.
Um den Kontrollfluss über eine Fragmentgrenze zu übertragen, wird eine explizite Nachricht von Fragment zu Fragment gesendet. Dazu wird eine sendende (*invoke*) Aktivität A in dem einen, sowie eine empfangende (*receive*) Aktivität B in einem anderen Fragment hinzugefügt. Diese zusätzlichen Aktivitäten sorgen dafür, dass das Ergebnis der Transition Condition von A richtig an B im zweiten Fragment übertragen wird. Da aber das Ergebnis der Transition Condition von A nicht direkt gelesen werden kann, wird um A ein Scope eingefügt (mit *supressJoinFailure=false*) der einen Fault Handler hat, dessen Aktivität im dem Fall, dass die Transition Condition *false* ist, aufgerufen wird und *false* an B sendet. Im *true* Fall sendet A (Aktivität im Scope) *true* an B.
Die Datenübertragung ist grundsätzlich wie die Aufteilung des Kontrollflusses aufgebaut. Zusätzlich zu dem *true* werden aber noch die eigentlichen Daten gesendet, mit dem *false* wird *null* gesendet. Im Falle eines Konfliktes von zwei eingehenden Datenlinks wird per Zufall entschieden. Dies ist nötig, um den häufigen Fall abzudecken, wenn es zwei alternative Pfade gibt, von denen aber immer nur einer navigiert wird. Das Überschreiben von vorhandenen Daten durch null wird durch eine Zusatzaktivität verhindert, welche die Einhaltung dieser Bedingung sicherstellt. Die konkrete Umsetzung in BPEL würde den Rahmen dieser Zusammenfassung sprengen, dafür verweisen wir hier auf das Paper.

Fazit

Dieser Ansatz ist interessant, weil er komplett auf vorhandenen Standards (BPEL, WSDL etc.) und Infrastruktur (BPEL-Maschine) basiert. Der fragmentierte Prozess verhält sich danach genauso wie ein normaler BPEL Prozess, nur dass er verteilt über mehrere Workflow Maschinen und Lokationen ablaufen kann.

2.2.2 B: A Novel Approach to Decentralized Workflow Enactment

Das Paper stellt einen neuen Ansatz vor, mit dem BPEL-Prozesse (zusätzlich zu bisher gängigen zentralen Workflow-Engines) auch dezentralisiert und verteilt ausgeführt werden können. Um die verteilte Ausführung zu ermöglichen, wird die Ausführungslogik in einzelne Aktivitäten umgewandelt, die ihre Koordination selbst übernehmen. Der Prozess wird in den Petri-Netz-Dialekt *Executable Workflow Networks (EWFN)* überführt, der dann ausgeführt werden kann. Die Nachrichten und Kontrollfluss-Anweisungen werden in einem Tuplespace abgelegt und können dort von den Teilnehmern abgerufen werden. Tuplespaces bilden als theoretisches Kommunikationsmodell die Grundlage der Sprache *Linda*.

Die Autoren stellen eine Vorgehensweise aus folgenden vier Schritten vor:

1. Process Modelling:
   Da gewünscht wird, dass der neue Ansatz das bisherige Vorgehen bei der Modellierung von Prozessmodellen nicht beeinträchtigt, wird in diesem Schritt nur ein gültiges, ausführbares BPEL-Modell erwartet.

2. Process Segmentation:
   In diesem Schritt wird das Prozessmodell auf die Teilnehmer (die physische Infrastruktur (Workflowmaschinen)) aufgeteilt und mit zusätzlichen Informationen versehen, in denen steht, wie welche Aktivitäten und Variablen auf die Teilnehmer verteilt werden – also ein Deployment Descriptor.
   Zur Aufteilung der Prozessmodelle werden zwei Alternativen vorgestellt:

   a. Manuelle Segmentierung
      Die Benutzer können die Segmentierung beliebig vornehmen.

   b. Automatische Segmentierung
      Die Segmentierung wird automatisch (basierend auf festgelegten Parametern) durchgeführt. Alternativ besteht die Möglichkeit, Kostenfunktionen zu verwenden, wodurch sich ein Optimierungs-Problem ergibt, das sich mithilfe von entsprechenden Algorithmen lösen lässt (Lokale Suche, Hill-Climbing, etc.).

3. Process Deployment
   Der Prozess wird umgewandelt in EWFN. Da EWFN auf Petri-Netzen basiert, hat es die gleichen Eigenschaften (keine zentrale Kontrollinstanz, Aktionen werden lokal ausgeführt und haben nur lokalen Effekt, asynchron). Die Autoren geben in Kapitel 3 eine formale Definition von EWFN an.
   Für die Umwandlung in EWFN wird jede BPEL-Aktivität als EWFN-Pattern dargestellt und wird über sogenannte *Interface Places* (quasi Schnittstellen) mit anderen Aktivitäten verbunden. Der Datenfluss innerhalb eines Patterns ist unabhängig vom Kontrollfluss. Daten eines Webservice-Aufrufs können also bereits vor einem Kontroll-Token erhalten werden. Die Daten werden dann zwischengespeichert und verarbeitet, sobald der Kontrollfluss das Pattern erreicht. Zusätzlich wird der originale BPEL-Prozess in XML vorgehalten, um eventuell Informationen nachzuschauen.

4. Distributed Process Execution
   Prozessmodelle werden per entsprechender receive- oder pick-Aktivität instanziiert, danach wird der Kontrollfluss entsprechend dem Modell ausgewertet. Kontrollflüsse innerhalb eines Segments werden auf einem lokalen Tuplespace ausgewertet, für Kontrollflüsse zwischen Segmenten wird der gemeinsame globale Tuplespace verwendet. Mögliche Operationen auf dem Tuplespace sind schreiben (*write*), lesen (*read*) und destruktiv holen (*take*).

Fazit

Dieses Paper ist zum einen interessant, da es zur Prozessmodellierung auf vorhandene Standards setzt (BPEL) zudem bietet die Umwandlung in Petri-Netze und die Verwendung von Tuplespaces auf jeden Fall eine sehr gute Basis für die verteilte Ausführung. Durch die Tuplespaces ist es nicht nötig, dass die einzelnen Fragmente (und auch Aktivitäten) direkt miteinander kommunizieren müssen, leider stellen diese auch das Bottleneck dieses Ansatzes dar.

2.2.3 C: Optimal Stratification of Transactions

Die *optimale Gliederung von Transaktionen* ist eine Methode, um eine optimale Verteilung, beispielsweise in Bezug auf Nebenläufigkeit und Antwortzeiten von Arbeitseinheiten, die auf

unterschiedlichen Prozessen deployed sind zu erreichen. Die Transaktionen mit ihren Eigenschaften sind essentiell für diesen Ansatz.

Der Ansatz kann im Outsourcing, Insourcing und Optimieren von BPEL-Prozessen sowie in der optimalen Ausnutzung von Ressourcen benutzt werden.

Modellierung:
Es wird ein Prozess modelliert, der wie ein Workflow aus atomaren Tasks, den *Basic Transactions* besteht. Der Gesamtprozess wird als *Global Transaction* bezeichnet. Innerhalb der *Global Transaction* werden die *Basic Transactions* durch Koordinationsprotokolle sowie das 2-Phasen-Commit-Protokoll (2PC) koordiniert.

Das 2PC-Protokoll ist aufgrund der hohen Abbruch-Wahrscheinlichkeit und dem ineffizienten Sperren von Ressourcen, die erst bei Beendigung des Protokolls wieder freigegeben werden, nicht brauchbar für lang laufende Transaktionen

Um die Performance der *Global Transactions* zu verbessern, wird eine Gliederung/Zusammenfassung von *Basic Transactions* genutzt. Die *Global Transaction* wird in kleinere Transaktionsgruppen gesplittet, die die nicht-funktionalen Anforderungen und Semantik der *Global Transaction* bereitstellen, während sie ihre Performance verbessern.

Diese Gruppen nennt man *Stratified Transactions*. Generell können mehrere *Stratified Transactions* in derselben *Global Transaction* enthalten sein. Jede *Stratified Transaction* wird charakterisiert durch ihre Ausführungszeit, Antwortzeit und Kosten. Die Kosten hängen stark an den Eigenschaften der benötigten Ressourcen. Zwei Eigenschaften von Ressourcen sind beispielsweise der Typ und der Ort der Ressource. Die Ressourcen werden unterschieden, da die Kosten variieren können. Beispielsweise sind die Synchronisationskosten von Ressourcen des gleichen Typs, sowie von Ressourcen, die am selben Ort liegen billiger als die von unterschiedlichen Ressourcen oder Ressourcen, die überall verteilt sind.

In der *Global Transaction* stehen die *Basic Transactions* aufgrund verschiedener Abhängigkeiten in Beziehung zueinander, beispielsweise Daten- und Kontrollabhängigkeiten. Hier wird angenommen, dass es eine Ordnung innerhalb der *Global Transaction* gibt, nach der die *Basic Transactions* ausgeführt werden. Diese partielle Ordnung über den *Basic Transactions* wird als Abhängigkeitsgraph bezeichnet. Dieser ist azyklisch.

Die *Basic Transactions* haben Eigenschaften, die betrachtet werden können. Beispielsweise die Aufruf- und Wiederherstellungskosten. Transaktionen mit hohen Aufruf- oder Wiederherstellungskosten sollten nicht mit Transaktionen, die eine hohe Abbruchwahrscheinlichkeit haben, kombiniert werden. Weitere Eigenschaften sind die Startzeit, die durchschnittliche oder maximale Ausführzeit und die Abbruch- oder Commit-Wahrscheinlichkeit.

In diesem Paper konzentriert man sich auf drei Eigenschaften:

- Kosten in Bezug auf die Ausführungszeit,
- Typ und
- Ort der benötigten Ressourcen.

Über den Abhängigkeitsgraphen wird ein so genannter *Stratification Graph* gesetzt. Die Idee eines *Stratification Graphs* ist es die *Basic Transactions* in disjunkte Teilmengen zu gliedern, z.B. Schichten, die die *Basic Transactions* mittels den 2PC-Protokoll koordinieren. Die einzelnen Schichten sind dann im *Stratification Graph* zusammengesetzt. Jede Schicht ist ein Knoten und es gibt eine Kante zwischen zwei Schichten wenn es mindestens ein Paar von *Basic Transactions* in den Schichten gibt, die im Abhängigkeitsgraphen miteinander verbunden sind. Jede Schicht hat eine persistente Nachrichten-Queue, die die eingehenden Nachrichten empfängt und bearbeitet. Eine Schicht kann nur ausgeführt werden, wenn sichergestellt ist, dass alle Abhängigkeiten der benutzten *Basic Transactions* erfüllt sind. Um Deadlocks zu vermeiden, muss der *Stratification Graph* auch azyklisch sein. Es gibt verschiedene Strategien um festzustellen ob der Graph azyklisch ist, diese sind analog zu denen aus der Deadlock-Erkennung und -Vermeidung.

Das Problem, die optimale Verteilung unter allen Möglichen zu finden, ist die Generalisierung des bekannten Problems der Graph-Optimierung und ist ebenso mindestens ein NP-Problem. Als Lösungsmöglichkeit wird hier ein gemischter Ansatz vorgeschlagen:

Als erstes wird der *Hill-Climbing*-Algorithmus angewendet, sobald die Suche ein lokales Optimum erreicht hat wird eine zweite Phase, basierend auf *Simulated Annealing* angestoßen, die es der Suche erlaubt das lokale Optimum wieder zu verlassen. Ein Kalibrierungsschritt wird benötigt um *Simulated Annealing* gemäß dem derzeitigen Stand der Suche zu konfigurieren.

Momentan wird der Verteilungsansatz zur Modellierungszeit angewendet. Zukünftige Arbeiten wollen den Ansatz so erweitern, dass er auch bei der Ausführung angewandt werden kann.

Fazit

Dieser Ansatz ist sehr interessant, da er als Eingabe nur ein Prozessmodell erwartet, das in Form von Transaktionen vorliegt. Zu den einzelnen Transaktionen müssen Informationen vorliegen über die Kosten, die sie verursachen um den *Stratification Graph* mithilfe von Graphoptimierungs-Algorithmen zu optimieren. Ansonsten sind keine Änderungen am Prozessmodell nötig.

2.2.4 D: BPEL Fragments for Modularized Reuse in Modeling BPEL Processes

Wenn Benutzer bisher einen neuen Geschäftsprozess erstellen wollten, mussten sie ihn komplett neu modellieren, auf Basis einer Vorlage oder eines Referenz-Prozesses aufbauen oder durch Veränderung eines bestehenden Prozesses weiterentwickeln. Die einfache und komfortable Wiederverwendbarkeit nach dem Baukastenprinzip wird bisher weder vom BPEL-Standard selbst noch von den aktuell verfügbaren BPEL-Erweiterungen unterstützt. In den bisherigen Ansätzen war es ausschließlich möglich, Prozessmodelle als Ganzes wiederzuverwenden. Benötigt wird also ein Container für wiederverwendbare Prozesslogik. Neben der Wiederverwendbarkeit können BPEL Fragmente dann auch gut für die verteilte Modellierung und Ausführung (somit auch Outsourcing) verwendet werden.
Das Auslagern des Prozessteiles als Webservices oder als Subprozess unter Verwendung von BPEL-SPE (sub-process extension) reicht nicht aus, da ein BPEL Fragment nicht ausführbar (kein gültiges BPEL) oder vollständig (z.B. Platzhalter in Vorlagen) sein kann, wenn beispielsweise im Fragment nur das Fault-Handling modelliert wird, welches dann als wiederverwendbarer Baustein an verschiedene Scopes angehängt werden kann.

Anforderungen an BPEL Fragmente

Folgende Anforderungen an BPEL Fragmente wurden in dem Paper identifiziert:

- Basic BPEL-Aktivitäten sind die kleinsten Einheiten eines Fragmentes und in jedem BPEL-Fragment muss mindestens eine davon vorhanden sein. Die zweite Anforderung wird später durch das Erlauben von *Opaque*-Aktivitäten etwas abgeschwächt.
- Ein Fragment muss in jeder Granularität möglich sein – seien es komplexe oder verschachtelte Aktivitäten, Scopes, einzelne Handler oder Kombinationen davon.
- Neben der Prozesslogik sollen auch Informationen über den Kontext (Variables, Partner Links, Correlation Definitions etc.) eines Prozesses / Scopes enthalten sein, was die Wiederverwendung vereinfacht.
- Außerdem soll dem Benutzer ermöglicht werden, *Variability Points* zu definieren, die als Platzhalter für Aktivitäten oder Attribute fungieren. Damit soll weiter ermöglicht werden, konkrete (nicht wiederverwendbare, weil z.B. domainspezifische) Daten und Abläufe aus dem Fragment auszulagern und so die Möglichkeiten der Wiederverwendung zu vergrößern.
- Der in BPEL definierte Kontrollfluss muss im Fragment enthalten sein / bleiben, da er die logischen Abhängigkeiten und die Ausführungsreihenfolge der Aktivitäten definiert.

Realisierung der Anforderungen

Im Paper wird eine formale Definition von BPEL-Fragmenten auf Basis des "Abstract BPEL" XML-Schemas gegeben, da es viele geforderten Eigenschaften (und die BPEL-Funktionalität) schon "von Haus aus" mitbringt. Neben einigen Details (Namespaces anpassen, Kardinalitäten ändern etc.) sind folgende Änderungen besonders hervorzuheben: Ein neues Konstrukt namens *<Fragment>* wird eingeführt, welches alle Attribute vom BPEL *<process>*-Tag erbt und das Wurzelelement eines BPEL-Fragmentes ist. An das Fragment darf jeweils einer der folgenden Elementtypen angehängt werden (nur einer, da sie sehr unterschiedliche Granularitäten haben):

- Aktivitäten: basic und structured
- Handler, im neue *<handlers>*-Konstrukt, das mehrere verschiedene oder gleiche Handler zusammenfassen kann
- Eine Gruppe von *<catch>* oder ein *<catchAll>*
- Events: onEvent (*onEventElements*), onAlarm (*onAlarmEventElements*), sowie das onAlarm (*onAlarmPickElements*) und onMessage (*onMessageElements*) der *<pick>*-Aktivität

Fazit

Dieser sehr interessante Ansatz folgt dem Web-Standards-Prinzip – nicht das Rad neu zu erfinden, sondern wiederzuverwenden – gleich doppelt: Einmal, indem er nicht viel Vorhandenes neudefiniert, sondern einfach auf Abstract BPEL aufbaut, und eine Metaebene höher, indem er es ermöglicht, BPEL-Fragmente in verschiedenen Prozessen wiederzuverwenden. Insgesamt eine praktische Ergänzung und Weiterentwicklung von Abstract BPEL.
Im Unterschied zu vielen anderen Ansätzen steht am Ende kein BPEL-Prozessmodell, sondern ein Fragment, aus dem ein Prozessmodell entstehen kann. Es werden also keine Antworten bezüglich paralleler oder verteilter Ausführung gegeben.

2.2.5 E: On Application of Structural Decomposition for Process Model Abstraction

Das Paper stellt eine Methode zum Abstrahieren von Prozessmodellen vor und legt dabei besonderes Augenmerk auf die strukturellen Gesichtspunkte. Abstraktion ist die Generalisierung eines Prozessmodells, wobei unwichtige Details weggelassen werden und am Ende ein weniger komplexes Prozessmodell entsteht. Der vorgestellte Abstraktionsansatz basiert auf *Process Structure Trees* (PST).

Für verschiedene Arten der Geschäftsprozess-Analyse werden zum Teil völlig unterschiedliche Perspektiven in unterschiedlichster Granularität ein und desselben Prozesses benötigt.
Der vorgestellte Ansatz kann auf beliebige als Graph darstellbare Prozessmodelle angewendet werden, beschränkt sich in diesem Paper allerdings auf die strukturellen Aspekte, also *wie* Elemente und nicht darauf *welche* Elemente abstrahiert werden können. Ein großer Teil des Papers widmet sich außerdem der Effizienz des Ansatzes.

Der Ansatz abstrahiert Aktivitäten, da diese (laut Untersuchungen) oft Gegenstand von Abstraktionen sind. Zur Darstellung der Prozess-Modelle verwenden die Autoren eine stark vereinfachte Notation anstelle von z.B. BPMN:

- Es gibt Aktivitäten und Gateways. Ein Gateway ist entweder ein *Split* oder *Join* und hat ein Kontrollfluss-Konstrukt (*and, xor, or*).
- Zwischen den Aktivitäten gibt es gerichtete Kanten, die den Kontrollfluss darstellen.
- Jede Aktivität hat mindestens eine eingehende und eine ausgehende Kante.
- Es gibt mindestens eine Start- und eine End-Aktivität, die je nur eine ein- bzw. ausgehende Kante haben.

Die Abstraktion soll in mehreren Schritten durchgeführt werden: In jedem dieser Schritte wird eine Aktivität aus einer Menge von unbedeutenden Aktivitäten abgearbeitet. Ausgabe eines solchen Schrittes ist ein neues abstrahiertes Prozessmodell, das als Eingabe für den nächsten Schritt verwendet wird.
Pro Abstraktionsschritt wird ein Element aggregiert, also mit anderen Elementen zusammengefasst.

An die Prozessmodelle wird die Anforderung gestellt, dass sie sicher sind (*Safeness*), also zu einem beliebigen Zeitpunkt eine Aktivität nicht mehr als einmal aktiv ist. Da durch die Abstraktion ein Prozessmodell stark verändert werden kann, ist es eine Voraussetzung, dass das zu abstrahierende Modell korrekt ist.

Auch an den entwickelten Ansatz werden Forderungen gestellt: Er darf die Reihenfolge der Aktivitäten durch die Abstraktion nicht verändern und die Abstraktionen müssen glatt verlaufen (*Abstraction Smoothness*). Die *Abstraction Smoothness* gibt an, wie viele Informationen in einem Abstraktions-Schritt verloren gehen. Je weniger Aktivitäten in einem Schritt generalisiert werden, desto glatter ist die Abstraktion.

Prozessmodelle werden für den Abstraktions-Algorithmus in *Canonical Single Entry Single Exit* (SESE)-Fragmente zerlegt. Ein solches Fragment hat genau eine eingehende und eine ausgehende Kante und kann somit durch eine aggregierte Aktivität ersetzt werden. Die vorher genannte Trennung zwischen Aktivitäten und Gateways findet hier keine Anwendung, da beide Arten einfach Knoten des Graphen sind.

Nachdem ein Prozessmodell in SESE-Fragmente aufgeteilt wurde und der dazugehörige *Process Structure Tree* (PST) erstellt wurde, kann die Abstraktion durchgeführt werden.

Für jede Aktivität wird das kleinste Fragment gesucht, das die jeweilige Aktivität und mindestens eine weitere Aktivität enthält. Dieses Fragment wird dann durch ein Aggregat ersetzt.

Im Weiteren untersuchen die Autoren die *Abstraction Smoothness*. Dazu werden verschiedene Prozessmodelle untersucht und mit zwei verschiedenen Szenarien abstrahiert. Das optimistische Szenario reduziert das Prozessmodell in jedem Schritt mit der kleinsten Anzahl an Aktivitäten; das pessimistische mit der größtmöglichen Anzahl. Die Autoren kommen zu dem Schluss, dass die Ergebnisse dann, wenn nur aggregiert werden kann, teilweise sehr schlecht sein können, also der Informationsverlust sehr hoch ist. Um diesem Problem entgegen zu wirken, wird das Eliminieren von Aktivitäten eingeführt. Wenn also das Fragment, das aggregiert werden soll, zu groß ist, kann die Aktivität stattdessen einfach eliminiert werden.

Fazit

Das Paper stellt einen Ansatz vor, der es leicht möglich macht ein Prozessmodell auf unterschiedlichen Detail-Ebenen zu abstrahieren. Ebenfalls interessant ist die Idee Aktivitäten einfach wegzulassen anstatt zu große Blöcke zu abstrahieren.
Auch wichtig ist, dass bei der Abstraktion sichergestellt wird, dass die *Ordering Constraints* des ursprünglichen Modells beibehalten werden.

2.2.6 F: A Fragment-driven Modeling Methodology

In diesem Paper wird ein Ansatz gesucht um Groupware mit der Prozessmodellierung zu fusionieren. Es wird ein kooperatives Prozessmodellierungskonzept bezüglich *Information Control Net* (ICN, Modellierungssprache) designed, in das die Echtzeit Groupware Funktionalität eingebettet wird. Dieser Ansatz ist der Vorschlag einer Prozessmodellierungsmethode, die komplett anders zu den klassischen Modellierungsansätzen ist. Da die Prozesse immer komplizierter werden und viele Firmen daran beteiligt sind, sollen mehr als ein Designer die Möglichkeit haben einen Prozess zu modellieren.

Bei dem Fragment-gesteuerten Ansatz wird eine logische hierarchische Prozessbeschreibungs-Methode konzipiert, in der alle Prozessfragmente von einem globalen Prozessmodellierungstool modelliert werden. Es gibt einen globalen Workspace, der durch Zugriffsberechtigungen kontrolliert wird. Das globale Prozessmodellierungstool stellt die Echtzeit Groupware Funktionalität zur Verfügung, so dass eine Gruppe von Designern zur selben Zeit an verschiedenen Orten gemeinsam einen globalen Prozess modellieren kann. Jede Firma muss nicht ihren kompletten internen Prozess preisgeben, sondern nur die Fragmente, die für die anderen interessant sind. Das globale Prozessmodell wird in einem Repository gespeichert, auf das die beteiligten Firmen Zugriff haben. Dieser Ansatz ist nicht nur in der Lage sichere globale Prozessmodellierung zu gewährleisten sondern auch eine auf Fragmentierung basierende, kooperative globale Prozessmodellierung.

Um in ICN Prozessmodellierung mit mehreren Unternehmen darstellen zu können, wird das ICN-Modell um die Begriffe globale Akteure, globale Gruppen und globale Firmen erweitert.

Die konzeptuelle Idee besteht auf einer Frontend-Komponente, die aus drei Hauptkomponenten besteht:

- Dem graphischen User Interface Manager: Er ermöglicht einer Gruppe von Designern einen gemeinschaftlichen globalen Prozess auf dem gemeinsamen Workspace zu definieren, bearbeiten, verifizieren und simulieren. Jeder Designer hat seinen eigenen Teil des Workspace, er kann die Fragmente eines anderen Designers lesen aber nicht bearbeiten.
- Die kooperierende Fragment-Client-Engine
- Dem lokalen Modellierungsdatenmanager: Er stellt die Gruppen Workflow Modellierungsfunktionalitäten wie Benutzerverwaltung, Zugriffskontrolle, Synchronisation und Eventhandling zur Verfügung.

Aus der operativen Sichtweise wird dieser Ansatz White-Box-Ansatz genannt, da alle Prozessfragmente die mit einem globalen Prozess verbunden sind an dem Ort, wo der Modellierer liegt genau beschrieben werden können. Diese können dann an die anderen Koordinatoren in Echtzeit verteilt werden. Dieser Ansatz verwendet das geschlossene e-Business Framework.

Abbildung 2 zeigt das operative Szenario des Ansatzes:

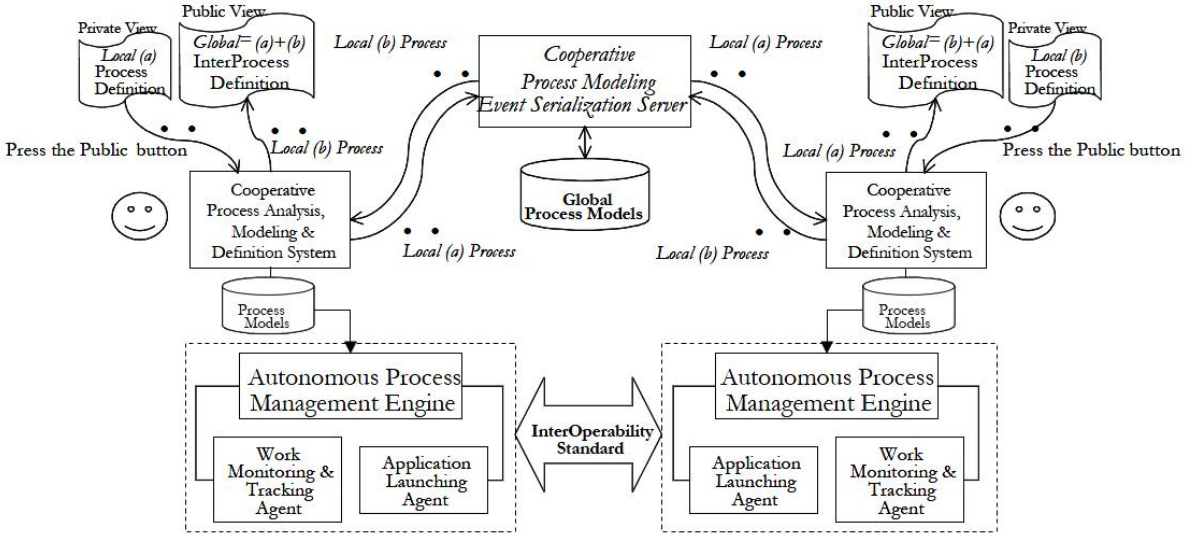

Abbildung 2: Operations-Szenario des Fragment-gesteuerten Ansatzes (Quelle: [6])

Dieser Ansatz passt sehr gut in ein System, das stark gekoppelte Interaktionen zwischen einer Gruppe von Menschen oder Firmen in Kooperation bereitstellt.

Fazit

Vorteile des Ansatzes:

- White-Box-Ansatz
- Mehrere kooperierende Designer können parallel an einem globalen Prozessmodell arbeiten

Man brauch keine Übersetzungsmaschinen entwickeln, die die Modellierungssprache der fragmentierten Prozesse in die Modellierungssprache des globalen Prozessmodells übersetzt, da die Modellierungssprache des globalen Prozessmodells WPDL/XPDL ist und die Prozessfragmente im globalen Repository gespeichert sind, welches das Interface zu den Runtime-Komponenten der einzelnen Firmen zur Verfügung stellt.

2.2.7 G: Modelling inter-organizational processes with process model fragments

Die Grundidee des Ansatzes ist es, Prozessfragmente zu erstellen, die so weit wie möglich unabhängig voneinander sind. Diese miteinander verbundenen Prozessfragmente stellen den organisationsübergreifenden Prozess dar. Jedes Prozessfragment gehört zu einer Organisationseinheit, welche für das Fragment verantwortlich ist. Das Fragment beinhaltet all die Aktivitäten, die die Organisationseinheit während der Ausführung des Prozesses ausführen muss. Die Organisationseinheit kann selbstständig das Fragment beschreiben und ausführen.

Um organisationsübergreifende Prozesse zu unterstützen, muss der Prozessfragmentierungsansatz folgende Bedingungen beachten:

- Jede teilnehmende Organisationseinheit muss in der Lage sein, seinen Teil des Prozesses selbstständig zu modellieren, analysieren und auszuführen. Ebenso muss eine Organisation ihren Teil vor den anderen verstecken können.
- Jedes Prozessfragment interagiert mit anderen Prozessfragmenten. Es benötigt Input-Dokumente, auf denen es arbeiten kann und produziert Output-Dokumente. Die Input-Dokumente werden von anderen Fragmenten bereitgestellt und die Output-Dokumente von anderen genutzt.
- Jede teilnehmende Organisation benutzt ihren eigenen Prozessmanagementansatz. Deswegen sollte der Prozessfragmentierungsansatz auf vielen verschiedenen Prozessmanagementansätzen nutzbar sein.

Prozessfragmente werden durch erweiterte Prozessmodelle beschrieben. Die Erweiterung beschreibt einerseits das Interface, das das Prozessfragment nutzt um mit anderen Fragmenten zu interagieren, andererseits hält es zusätzliche Informationen zur Ausführung und Analyse des Fragments bereit. Das Interface ist öffentlich und wird von anderen Organisationen genutzt um Fragmente miteinander zu verbinden. Die Abfolge der Aktivitäten dagegen ist privat.

Die Unterteilung eines Prozesses in Fragmente basiert auf dem Kontrollfluss und dem Datenfluss. Der Kontrollfluss ist die Abfolge der Aktivitäten und der Datenfluss beschreibt die Dokumente, die die Aktivitäten benötigen und produzieren. In organisationsübergreifenden Prozessen existieren fragmentübergreifende Kontroll- und Datenflüsse. Die Dokumente die zwischen Aktivitäten innerhalb eines Prozessfragments übergeben werden haben keinen Einfluss auf den fragmentübergreifenden Datenfluss. Das Hauptproblem ist es, die Struktur und Bedeutung eines Dokumentes auf beiden Seiten zu beschreiben. Um dies zu tun wird die externe *Document Type Description Language* (DTDL) verwendet.

Ein organisationsübergreifender Prozess besteht aus mehreren Prozessfragmenten. Die Beschreibung der Prozesse besteht aus Prozessmodellfragmenten, die miteinander verbunden werden müssen um den Kontroll- und Datenfluss zwischen den Fragmente zu beschreiben. Zur Buildtime muss keine so komplexe Beschreibung vorhanden sein, da manchmal nicht alle Fragmentinformationen, vor allem die Fragmente selbst, bekannt sind. Deswegen muss es möglich sein die Prozessbeschreibung während der Ausführung zu vervollständigen. Es ist möglich Interfaces beim Start der Ausführung nicht verbunden zu lassen. Während der Ausführung muss die Prozessmaschine das nicht verbundene Interface finden und den Prozessmodellierer darüber informieren.

Es gibt zwei Arten Interfaces miteinander zu verbinden:

- Vertikale Verbindung: Diese Verbindung erlaubt die hierarchische Verfeinerung von Aktivitäten in einem Vater-Fragment von Kind-Fragmenten. Ein Prozessfragment kann mehr als eine vertikale Verbindung haben.
- Horizontale Verbindung: Dieser Typ verbindet ein Dokument mit einem Event. Über die kann die Weitergabe von Events und Dokumenten beschrieben werden.

Die Verbindung von Prozessfragmenten ist ein Frageprozess. Die Organisation, die ihre Prozessfragmente verbinden möchte muss sicherstellen, dass die Interfaces zusammen passen. Jede Organisation speichert die Verbindungen lokal, dies ist nötig damit die Unabhängigkeitsanforderungen erfüllt sind. Mit der Prozessmodellierungsfragmentierung kann jeder Organisation ihren Teil des organisationsübergreifenden Prozesses beschreiben, unabhängig von den anderen Organisationen. Nur die Beschreibungen der Interfaces sind öffentlich. So werden die am Anfang beschriebenen Bedingungen erfüllt. Das Metamodell ist unabhängig von einem bereits existierenden Prozessmodellansatz, so ist es möglich es in unterschiedlichen Workflowsystemen zu implementieren.

Fazit

Dieses Paper stellt einen Ansatz vor, der sehr auf Unternehmensbedürfnisse angepasst ist. Jede Unternehmenseinheit kann ihre Fragmente verwalten und unabhängig bearbeiten. Ebenso muss nicht der Inhalt des Fragmentes offen gelegt werden sonder nur die Schnittstelle veröffentlicht werden.

2.2.8 H: Supporting Workflow Cooperation within and across Organizations

Das Paper stellt einen Ansatz vor, der die Zusammenarbeit von verschiedenen Geschäftsprozessen auf Basis von Events realisiert.

Die Kooperation zwischen unterschiedlichen Prozessen verschiedener Organisationen wird erschwert durch die Ungleichheit der eingesetzten Plattformen. Der in diesem Paper vorgestellte Ansatz realisiert die Kooperation mithilfe eines Publish/Subscribe-Event-Services, der leicht auf bestehende Systeme aufgesetzt werden kann. Events können Parameter beinhalten, über die Variablen-Werte vom Sender an den Empfänger übergeben werden können. Als Event-Service wählten die Autoren *Distributed Eventbased Reactive Processing Architecture* (*DERPA*) und erweiterten diesen um ihn an ihre Anforderungen anzupassen.
Es wird ein Model für den Ansatz vorgestellt sowie ein System, das das Model implementiert.

Im Modell werden Punkte an denen Interaktionen mit anderen Workflows stattfinden sollen mit *event nodes* definiert. Diese können als normale Workflow-Aktivitäten eingebunden werden. Interaktionen können das Senden und Empfangen von Events (also auch Synchronisation und Datenaustausch), Benachrichtigungen über Status-Änderungen oder Service-Execution-Anfragen sein. *Event nodes* können entweder vom Typ *send (produce)* oder vom Typ *request (consume)* sein. *Send nodes* sind nicht-blockierend, *request nodes* sind blockierend.

Jedes Event gehört zu einer Event-Klasse, die Events mit gleicher Semantik gruppieren. Eine Event-Klasse hat einen eindeutigen Namen und eine beliebige Anzahl an Parametern, die vom Typ integer, string, real, boolean, datetime, record oder ein Byte-Stream sein können.
Send nodes haben einen Namen und Parameter, die durch ein 2-Tupel *(Name, Wert)* definiert werden. Werte können Konstanten *(Name, "Wert")* oder auch Namen von Workflow-Variablen

(Name, Variablen-Name) sein.
Request nodes empfangen immer mindestens eine Event-Instanz und nur solche, die nach dem *request* erzeugt werden. *Request nodes* haben einen eindeutigen Namen, eine *request expression*, die die Art der Events spezifiziert die der Knoten empfangen soll und Parameter, die angeben, welche Werte gespeichert werden sollen.

Im Weiteren wird noch eine Implementierung des DERPA Event-Services vorgestellt.

Fazit

Dieses Paper stellt eine Möglichkeit vor, wie verschiedene Prozesse untereinander über Events kommunizieren können.

Leider enthält dieses Paper nicht, wie zuerst gedacht, einen Fragmentierungsansatz und wird deshalb in der weiteren Fachstudie nicht betrachtet.

2.2.9 I: Decentralizing Execution of Composite Webservices

Webservices sind funktionale Einheiten, welche durch BPEL zu zusammengesetzten Webservices aggregiert (orchestriert) werden. Ein BPEL-Prozess steht selbst wieder als Webservice zur Verfügung (rekursives Aggregationsmodell). Auch wenn die orchestrierten Webservices für sich verteilt sind (jeder Webservice kann auf einem anderen Rechner oder einem ganzen Cluster laufen), so ist mit BPEL die Kontrolle und Navigation des Prozesses zentral. Ziel dieses Ansatzes ist es, solche Probleme, die durch die zentralisierte Kontrolle auftreten (Bottleneck, Skalierbarkeit, Single Point of Failure, etc.), zu beseitigen, indem das Prozessmodell anhand der Datenabhängigkeiten partitioniert wird. Aus einer einzelnen BPEL Datei werden automatisch mehrere dezentralisierte Prozesse gemacht, dabei sollen die Kommunikationskosten minimiert und der Durchsatz maximiert werden.

Grundidee

Der sogenannte Glue-Code, also das was einen Workflow ausmacht (BPEL-Code der die Webservices orchestriert), muss nicht notwendigerweise zentral ablaufen, sondern kann auch auf die jeweiligen Server mit den Webservices ausgelagert werden. Die Daten erzeugenden Prozess-Fragmente können diese sofort an die Fragmente senden, welche die Daten benötigen und weiterverarbeiten. Das setzt jeweils eine lokale BPEL Maschine voraus, was aber angenommen werden kann, da viele Webservices selbst intern wieder nur BPEL Prozesse sind. Als Ergebnis erhält man eine dezentralisierte Ausführung, welche die Parallelität (durch den verteilten und parallel ausgeführten Glue-Code) erhöht und die Anzahl der Nachrichten verringert, da die Webservices nicht dem zentralen Koordinator antworten, sondern die Daten direkt weiterschicken können.

Partitionierung

Jedes BPEL Konstrukt wird als einzelner Knoten einem Datenabhängigkeits-Graphen repräsentiert, wobei es portable Knoten gibt, die auf einem beliebigen Server ausgeführt werden können, und feste Knoten, die auf einem bestimmten Server ausgeführt werden müssen, da dort beispielsweise benötigte Daten vorliegen.
Es gibt verschiedene Ansätze diesen Graphen zu partitionieren, darunter auch Algorithmen die zur Partitionierung von Programmen für Multiprozessor-Umgebungen entwickelt wurden. Diese versuchen einzelne Knoten zu Fragmenten zusammenzulegen (*merge*), um den Overhead – trotz

möglicher Einbußen bei der Parallelität – zu minimieren. Diese lassen sich aber nicht direkt anwenden, da zusätzliche Einschränkungen beachtet werden müssen (feste <-> portable Knoten, Durchsatz als Optimierungskriterium und nicht Ausführungszeit), aber die Ideen der Algorithmen werden verwendet. In den Algorithmen können nur benachbarte Knoten zusammengelegt werden, was insofern nicht ideal ist, da der Kommunikations-Overhead bei Workflows dann deutlich abnimmt, wenn parallel zur Datenabhängigkeit zusammengelegt wird. Dadurch sinkt zwangsläufig die Anzahl der Nachrichten, die gesendet werden müssen (da die Nachricht zwischen den beiden Knoten aufgrund der gemeinsamen Ausführung entfällt). Um dies zu gewährleisten, müssen die Knoten umsortiert werden, was der Ansatz genauso wie die Validierung dieser Neuanordnung im Paper näher beschreibt.
Es gibt exponentiell viele Möglichkeiten diesen Graphen zu fragmentieren, weshalb ein alle Möglichkeiten durchprobierender Algorithmus nicht sinnvoll ist. In der Folge beschreibt das Paper die *merge-by-def-use*-Heuristik, welche die Fragmentierung so durchführt, dass der Durchsatz maximiert wird. Die dem zugrunde liegende Idee ist, dass die portablen Knoten, die auf dem Graphen (mit den Datenabhängigkeiten) liegen, zwischen zwei festen Knoten beliebig zwischen den beiden Fragmenten verschoben werden können. *def-use* ist die Beziehung zwischen dem Knoten, auf dem ein Datum generiert und dem, wo es konsumiert wird – also die einfache Datenabhängigkeit. Diese Heuristik wird durch zwei weitere Heuristiken (*greedy-merge* und *pooling*) weiter optimiert.

Des Weiteren gibt das Paper ein Kostenmodell für die Optimierung des Durchsatzes an, genauso wie die Ergebnisse experimenteller Tests der Laufzeit und Compilezeit (Dauer der Partitionierung) des Ansatzes und der verschiedenen Heuristiken.

Fazit

Dem Ansatz kann man positiv anrechnen, dass in ihm nicht die Partitionierung eines Plain-BPEL Prozesses im Mittelpunkt steht, sondern die einer Menge von Webservices, welche durch BPEL orchestriert werden. Aufgrund dieser anderen Sichtweise geht es hier darum, wie man den Glue-Code zu den Webservices hin verteilt (also um den Webservice ein Fragment baut) und nicht wie man komplett neue Fragmente generiert.
In diesem Zusammenhang ist es auch gut den Durchsatz zu optimieren, da Middleware generell auf Durchsatz getrimmt ist.
Fraglich bleibt, ob die Server auf denen die Webservices laufen überhaupt zusätzlichen BPEL Code ausführen können (Kapazität) und dürfen (weil der Webservice z.B. in einer anderen Firma läuft). Im Extremfall würde das bedeuten, dass ein sehr beliebter Webservice noch Glue-Code von hunderten Workflows ausführen muss, die ihn aufrufen. Diese praktischen Probleme werden im Paper nicht betrachtet, weisen aber darauf hin, dass der Ansatz eher für den organisationsinternen Gebrauch einsetzbar ist. Umgehen ließen sich diese Probleme, indem externe Webservices in einen internen gekapselt werden, der dann auch Teile des Glue-Codes ausführen kann, oder indem man das *invoke* manuell zu einem festen Knoten auf einem internen Server erklärt, was allerdings die Flexibilität und Freiheiten des Algorithmus einschränken würde.

2.2.10 J: Distributed Workflow Execution based on fragmentation of Petri Nets

Aufgrund wirtschaftlicher Veränderungen steigt die Nachfrage nach der geografischen und organisatorischen Verteilung von Workflows auf verschiede Orte. Dieses Paper schildert einen Ansatz mit dem auf Petri-Netzen basierte Workflows, unter Einsatz von Fragmentierungs- und Replikations-Mechanismen aus dem Gebiet der relationalen Datenbanken, fragmentiert werden können. Dazu

werden *predicate/transition nets* (PrT-nets), eine Petri-Netz-Abwandlung eingesetzt.
Die Fragmentierung findet zur Modellierungszeit statt und soll explizit die verteilte Ausführung des Prozessmodells ermöglichen. In Paper [K] wird ein Verfahren beschrieben, welches ebenfalls Petri-Netze verwendet, aber zur Laufzeit angewendet wird.

Existierende Konzepte

Von relationalen Datenbanken ist die horizontale (Verteilung der Datenbankzeilen) und vertikale Fragmentierung (Verteilung der Spalten) bekannt. Der Overhead der durch diese Verfahren entsteht wird durch die Steigerung der Zuverlässigkeit und Parallelität kompensiert.

Zur Modellierung von Workflows gibt es viele unterschiedliche Sprachen. Petri-Netze haben den Vorteil, dass sie einerseits eine grafische Repräsentation, auf der anderen Seite aber eine weithin genutzte formale Definition besitzen. Ein PrT-Netz ist ein Petri-Netz, dessen Stellen (*Predicates*) ein relationales Datenbankschema darstellen. Die Marken auf den Stellen repräsentieren Relationen. Eine Transition (dargestellt durch einen Pfeil zwischen zwei Stellen) ist eine Operation mit den Marken (Relationen) der angrenzenden Stellen. PrT-Netze erlauben es logische Bedingungen (Transition Conditions in BPEL) für das Feuern einer Transition zu definieren.

Patterns & Fragmentierungsansatz

Im Paper werden 4 einfache Prozess-Patterns beschrieben, aus denen man komplexe Prozesse zusammenbauen kann. Für diese Patterns werden später Methoden beschrieben, wie man sie fragmentieren kann. Wenn ein Prozessmodell nur aus diesen Patterns aufgebaut ist (oder in sie zerlegt werden kann), definiert der Ansatz somit eine Fragmentierung.

- Sequenz (Hintereinander-Ausführung)
- Alternative (konkurrieren um dieselben Ressourcen; mindestens eine gemeinsame Start- und/oder End-Stelle)
- Parallelausführung (Ausführung unabhängig möglich; keine gemeinsamen Stellen)
- Wiederholung (sind voneinander abhängig)

Ein Netz wird fragmentiert, indem die Stellen geteilt werden. Eine Stelle durch die eine Fragment-Grenze läuft wird vervielfacht und jeweils eine Kopie (Schnittstelle genannt) in jedes beteiligte Fragment gelegt. Bei der Replikation dürfen nur komplette Fragmente repliziert / verteilt werden, nicht Teile eines Fragmentes.

Auch dürfen Workflows nicht beliebig fragmentiert werden, sondern müssen drei (aus der Fragmentierung von Datenbanken bekannten) Bedingungen gehorchen:

- Vollständig: Jede Aktivität eines Prozesses ist mindestens in einem Fragment.
- Minimal: Kein Fragment enthält Aktivitäten die nicht auch schon im ursprünglichen Netz enthalten waren.
- Nicht überlappend: Keine Aktivität ist in mehreren Fragmenten, außer die Überlappung ist für die Rekonstruktion des Original-Netzes nötig (also die Aufteilung an den Fragment-Grenzen).

Kommunikation der Fragmente

Die Zuweisung der Marken (Relationen) auf die einzelnen Schnittstellen kann, abhängig vom Prozessmodell, durch Replikation und Fragmentierung geschehen.

- Bei der Replikation werden die Marken (und ihre Veränderungen) aus einem Fragment zu allen anderen Fragmenten repliziert, welche die Stelle enthalten. Dadurch ist es möglich für jede Transition lokal zu prüfen ob sie feuern kann.
- Bei manchen Stellen reicht jedoch eine horizontale Fragmentierung aus, beispielsweise bei einer Alternative, bei der jeder Pfad ein eigenes Fragment ist. Hier kann das Netz so verändert werden das die entsprechenden Marken nur an das Fragment übertragen wird (praktisch ohne Zeitdruck, anders als bei der Replikation), welches später die Ausführung durchführen wird. Für die Schnittstellen in den anderen Fragmenten ist die Relation uninteressant.

Methoden zur Fragmentierung der Patterns

Das Paper enthält drei Methoden, die beschreiben wie man die einzelnen Patterns fragmentieren kann. Somit lässt sich ein Prozessmodell Schritt für Schritt fragmentieren.

- Vertikale Fragmentierung: Ein Netz wird in zwei vertikale Fragmente aufgeteilt, wenn Fragment B von Fragment A abhängig ist (das heißt mindestens eine Transition läuft über die Fragment-Grenze von A nach B), aber nicht umgekehrt. Dies deckt das o.g. Sequenz-Pattern ab.
- Horizontale Fragmentierung: Ein Netz wird horizontal fragmentiert wenn kein Fragment auf dem anderen basiert. Das heißt Alternativen (gemeinsame Ausgangs- und/oder End-Schnittstelle) und Parallelausführungen (keinerlei gemeinsame Schnittstellen) können horizontal fragmentiert werden.
- Diagonale Fragmentierung: Wenn zwei Fragmente gegenseitig voneinander abhängig sind, ist das Netz diagonal fragmentierbar. Das heißt bildlich gesprochen, dass man den Kreis der Abhängigkeiten durchschneidet, so dass jedes Fragment einen Halbkreis abdeckt.

Fazit

Dieser formale Ansatz sorgt für Beweisbarkeit, ist aber auch nicht so flexibel wie andere Ansätze: Der Prozessmodellierer kann beispielsweise die Zuteilung der Aktivitäten zu den Fragmenten nicht beliebig nach den Business Anforderungen durchführen, sondern ist auf die Granularität der Patterns eingeschränkt. Die Zusammensetzung aus Patterns hat jedoch den Vorteil, dass die Korrektheit von vornherein, während der Modellierung erhalten wird. Das Prozessmodell muss als Petri-Netz vorliegen, wobei eine Transformation von BPEL dahin bekannt ist.
Keine Antwort wird auf die Frage geliefert wie die ideale Fragmentierung gefunden werden kann, gerade in Bezug auf organisatorische Grenzen oder Nähe zu Ressourcen oder Arbeitsplätzen. Unbeantwortet bleibt auch wie der Nachrichtenaustausch konkret aussehen kann.

2.2.11 K: Dynamic workflow model fragmentation for distributed execution

In diesem Paper wird ein Ansatz zur dynamischen Fragmentierung von Workflows mithilfe von Petri-Netzen vorgestellt. Die Autoren legen den Schwerpunkt dabei nicht auf Design und Implementierung,

sondern auf formale Methoden. Die Fragmentierung erfolgt Schritt für Schritt während der Ausführung des Workflows.

Basis des Ansatzes bildet das zentralisierte, azyklische Workflow-Modell *WF-net*. WF-net ist eine bestimmte Klasse von Petri-Netzen und wird als Tupel *(T, P, A)* dargestellt, wobei *P* Places, *T* Transitions und *A* Arcs sind. WF-net ist außerdem *well-structured*, alternative Flüsse werden also immer mit der logischen Verknüpfung wieder zusammengeführt, mit der sie getrennt wurden.
Die Autoren unterscheiden zwischen statischer Fragmentierung, bei der die Fragmente vor der Ausführung festgelegt werden und der in diesem Ansatz verwendeten dynamischer Fragmentierung, die zur Laufzeit erfolgt:

Nach jedem ausgeführten Task wird aus dem verbleibenden Teil des Modells ein neues Fragment erzeugt, bei einem AND-Split werden entsprechend mehrere Fragmente erzeugt. Diese Fragmente können dann auf verschiedene Server verteilt und ausgeführt werden.
Die Fragmentierung erfolgt so, dass jedes Fragment genau eine Quell-Transaktion hat und damit auch nur genau eine Aufgabe zu erfüllen hat.
Bei einem AND-Split werden die Fragmente derart erzeugt, dass nur eines der Fragmente den Teil des Modells nach dem zum Split gehörenden AND-Join enthält.

Die Autoren untersuchen im Weiteren noch die Korrektheit des vorgestellten Ansatzes und stellen fest:

- Die Fragmentierung ist vollständig: keine Informationen des originalen Workflows gehen verloren.
- Jedes Fragment ist vollständig: Jedes Fragment hat alle Informationen, die es zur Ausführung benötigt.
- Verhaltensgleichheit: Die Fragmente haben die gleichen Verhaltens-Charakteristika wie das Fragment, aus dem sie erzeugt wurden.

Fazit

Das Paper stellt einen interessanten Ansatz vor, der die Fragmentierung von Petri-Netzen zur Laufzeit ermöglicht. Der Ansatz ermöglicht es, das Prozessmodell zur Laufzeit dynamisch auf einer Maschine auszuführen, die die benötigten Daten bereithält, was es unnötig macht größere Datenmengen zu transferieren. Durch das Aufteilen einzelner Aktivitäten auf unterschiedliche Server nach einem Split kann auch echte Parallelität bei der Ausführung erreicht werden.

2.2.12 L: Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition

Dieses Paper beschreibt einen Ansatz, bei dem ein Workflows in eine Hierarchie (Baum) von SESE-Fragmenten (*Single-Entry-Single-Exit*) zerlegt wird. Vorteil dieser SESE-Fragmente ist, dass sie isoliert analysiert und bearbeitet werden können, was aufgrund der kleineren Größe deutlich schneller durchführbar ist. Als ein zweiter Schritt werden Heuristiken beschrieben, welche diese Analyse schnell durchführen können und zusätzlich die Fehler so lokalisieren können, dass eine leichte Behebung der gefundenen Probleme möglich ist.
Die Fragmentierung in SESE-Fragmente hat aber vielfältige andere Anwendungsmöglichkeiten neben

der Kontrollflussanalyse von Workflows. Auch für die Umwandlung von einer Graphen-basierten Workflows (z.B. BPMN oder UML-Aktivitätendiagramme) in Block-basierte (z.B. BPEL) ist die SESE-Fragmentierung entscheidend [13]. SESE-Fragmente helfen beim besseren Verstehen von komplexen Workflows, durch SESE Fragmentierung und Betrachtung der einzelnen Fragmente [13], genauso wie beim Vergleich zweier ähnlicher, aber unterschiedlicher Workflows. Jussi Vanhatalo et al. beschreiben in *AutomaticWorkflow Graph Refactoring and Completion* [14] die Verwendung beim Refactoring und der Vervollständigung von Workflows.

Aufteilung des Workflows in SESE-Fragmente

Diese Fragmentierung bringt Vorteile, weil anschließend die Fragmente isoliert betrachtet werden und nachfolgende Schritte wie z.B. die heuristische Prüfung der Intaktheit schneller ausgeführt werden können.
Auf einen Algorithmus der die Fragmentierung in Linearzeit durchführt, wird im Paper verwiesen. Grundsätzlich gilt, dass zwei Fragmente sich nicht überlappen, aber ineinander geschachtelt sein können. Aus den Fragmenten eines Workflows kann man einen Strukturbaum bilden, dessen Wurzel der Gesamtprozess ist und dessen Blätter Fragmente sind, in die kein anderes Fragment verschachtelt ist. Dieser Strukturbaum wird im nächsten Schritt von den Heuristiken verwendet.

Heuristiken, die die Intaktheit (en: *soundness*) von Fragmenten prüfen

Die Heuristiken zur Ausführung vor den *complete techniques* gedacht, weil sie die Intaktheit (oder das Gegenteil), bei der Mehrzahl der Fragmenten, in linearer Zeit beweisen können. Ob ein Prozessmodell intakt ist, kann geprüft werden, indem im Strukturbaum von den Blättern nach oben jedes Fragment isoliert auf seine Intaktheit geprüft wird.
Zur Durchführung werden heuristische Regeln angegeben, die durch die Erkennung einfacher Strukturen in (oft kleinen) Fragmenten, Aussagen über die Intaktheit oder Nicht-Intaktheit eines Fragments geben. Beispielweise ist ein Fragment intakt, wenn es genau eine *decision*-Aktivität am Anfang, eine *merge*-Aktivität am Ende und dazwischen keine *fork*- oder *join*-Aktivität hat. Nicht intakt ist es beispielsweise, wenn es eine oder mehrere *decision*-Aktivitäten, aber keine *merge*-Aktivität im Strukturbaum hat.

Fazit

Interessant anzumerken ist die Untersuchung von 340 Prozessmodellen aus der Industrie, deren Ergebnisse im Paper enthalten sind. Im Schnitt dauerte die Analyse einer Prozessmodells weniger als 0,1 Sekunden (auf einem normalen IBM-Notebook), was von der Analysedauer her das Ziel des Ansatzes erfüllt, die Analyse während der Modellierung regelmäßig durchzuführen.
Die SESE-Fragmentierung ist hier nur ein Tool um die Analyse von Workflows zu verbessern, aber die Fragmente können, wie oben beschrieben, auch vielfältig anderweitig verwendet werden.

2.3 Ausgewählte Ansätze

Warum wir diese vier Ansätze für eine genauere Betrachtung ausgewählt haben, wollen wir im Folgenden jeweils kurz begründen:

2.3.1 A: Role-based Decomposition of Business Processes using BPEL

Bei diesem Ansatz ist interessant, dass die konkrete Ausführung betrachtet wird. Er beschreibt wie einzelne Fragmente (auch über Workflow-Maschinen-Grenzen hinweg) nicht mit besonderen Konstrukten als ein Workflow agieren, sondern mit Standard-BPEL und Webservices.
Außerdem ist dies der einzige Ansatz, der es dem Prozessmodellierer erlaubt, die Aktivitäten ausschließlich nach Business-Anforderungen (Stichwort: Outsourcing) anzuordnen und die dazu nötigen fragmentiert ausführbaren Prozessmodelle (eines für jedes Fragment) dann automatisch erstellen zu lassen.

2.3.2 B: A Novel Approach to Decentralized Workflow Enactment

Dieser Ansatz ist interessant, da er im Vergleich zu anderen Papern einen ganz anderen Ansatz zur Kommunikation zwischen den Fragmenten verwendet, der eine direkte Kommunikation zwischen den einzelnen Aktivitäten unnötig macht.
Durch die Verwendung von Tuplespaces ist es auch nicht nötig, dass die einzelnen Fragmente direkt miteinander kommunizieren. Es muss nur der Tuplespace bekannt sein. Dadurch ist es auch möglich, dass die Aktivitäten sich selbständig koordinieren.
Ein weiterer interessanter Gesichtspunkt ist, dass das Verfahren es ermöglicht die Ausführungslogik auf der Maschine auszuführen, auf der auch der entsprechende Service liegt. Dadurch wird ein übermäßiges Senden von Daten verhindert und führt so zu einer effizienteren Ausführung des Prozesses.

2.3.3 C: Optimal Stratification of Transactions

Wir haben uns dazu entschieden dieses Paper näher zu betrachten, da dieser Ansatz aufgrund seiner Flexibilität in der Anwendung sehr interessant ist. Er kann auf jedes Prozessmodell, das in Form von Transaktionen vorliegt angewendet werden. Es wird ein Algorithmus vorgestellt, der es ermöglicht anhand der Kosten, die eine Transaktion verursacht eine optimale Verteilung zu finden. Dieser Algorithmus besteht aus bekannten Graphoptimierungsalgorithmen.
Allerdings kann nur ein vereinfachter Kontrollfluss, der aus SPLIT und AND-JOINS besteht verwendet werden. Die soll aber in zukünftigen Arbeiten verbessert werden.

2.3.4 I: Decentralizing Execution of Composite Webservices

Dieser Ansatz trägt dem Fakt Rechnung, dass BPEL-Prozesse normalerweise mehrere Webservices verwenden und diese oft schon auf unterschiedlichen Servern laufen. Da ist es naheliegend Teile des Prozesses zu den Webservices zu "schieben". Außerdem können die Server, welche die Webservices bereitstellen, auch direkt untereinander kommunizieren, anstatt die Ergebnisse immer zurück zum Start-Fragment zu senden. Auch für die Umsetzung dieses Ansatzes ist keine Erweiterungen der Workflow Standards und Middleware erforderlich.
Da diese Ideen sehr viel versprechend klingen, haben wir uns entschieden diesen Ansatz näher zu untersuchen.

3 Kriterienkatalog

Zur Bewertung und zum Vergleich der einzelnen Papers wurde ein Kriterienkatalog mit insgesamt 16 Kriterien zusammengestellt, der alle wichtigen Fragestellungen zur Art und Funktionsweise der Ansätze enthält:

3.1 Grundlagen

1. Was wird unter einem Fragment verstanden und wie wird es definiert?
2. Was ist die Granularität der Fragmentierung?
    - Ist eine Aktivität, ein Scope oder ein *Single-Enty-Single-Exit*-Block, etc. das kleinste Granual in einem Fragment?
3. Wieso wird fragmentiert? Was sind die Ziele und Motivationen des Ansatzes?
    - Optimierung zur Einhaltung nichtfunktionalen Anforderungen (QoS) z.B. Durchsatz, Kosten, Antwortzeit
    - Unterstützung der Modellierung z.B. Wiederverwendung
    - Analyse, z.B. Soundness-Check

3.2 Wie wird im Ansatz fragmentiert?

4. Nach welchen Kriterien werden die Aktivitäten auf die Fragmente verteilt?
   Findet dies automatisiert oder manuell statt?
    - „Von Hand“, manuell durch den Modellierer
    - Automatisch mit Kostenfunktionen: Durchsatz, Ausführungszeit, Overhead, Kommunikationskosten, Ressourcenlokation, etc.
    - Automatisch, "wie der Algorithmus es will", Kriterien im Algorithmus liegend
    - Falls auf die Identifikation der Fragment nicht eingegangen wird: Was gäbe es für Möglichkeiten zur Identifikation, die zu diesem Ansatz passen würden?
5. Wird die Fragmentierung zur Design- oder Laufzeit durchgeführt?
6. Wie werden BPEL-Konstrukte unterstützt? (Nur falls BPEL als Eingabe vorgesehen)
    - Welche BPEL-Version wird unterstützt/vorausgesetzt
    - Wie wird mit Exception Handling und Compensation umgegangen?
    - Funktioniert Dead-Path-Elimination (auch über Fragmentgrenzen hinweg)?
    - Wie wird Correlation unterstützt?

3.3 Anwendbarkeit des Ansatzes

7. Was für ein Prozessmodell wird als Eingabe verwendet?
    - BPEL, BPEL-D, Petri-Netz, BPMN, etc.
8. Welche zusätzlichen Änderungen am Prozessmodell sind vor der Fragmentierung notwendig?
    - Keine Änderung, Einfügen von Hilfs-Konstrukten, Umwandlung in Petri-Netz, etc.
9. Können vorhandene Standard-Modellierungs-Tools (beispielsweise für BPEL oder BPMN) weiter genutzt werden?
10. Können bestehende, bereits modellierte Prozessmodelle fragmentiert werden?
11. Ist die Wiederverwendung der Fragmente vorgesehen?

3.4 Ausführbarkeit des Ansatzes (falls zutreffend)

12. Ist zur Ausführung des fragmentierten Prozesses eine normale BPEL-Engine ausreichend, oder werden zusätzliche Komponenten oder Erweiterungen benötigt?
13. Wie findet die Kommunikation zwischen den Fragmenten statt?
    - Messaging, BPEL Webservice, Tupelspace, etc.
14. Wie groß ist der Overhead der Fragmentierung?
    - Sei es durch die Kommunikation oder durch das Hinzufügen von Hilfs-Konstrukten (höhere Anzahl der Elemente die navigiert werden müssen).
15. Wird der Workflow von einem zentralen Koordinator kontrolliert? Kann der globale Zustand des Workflows eingesehen werden, oder ist dieser über die Fragmente verteilt?
16. Wird nur die verteilte Ausführung von Fragmenten beschrieben, oder auch wie eine optimale Fragmentierung gefunden wird?

4 Bewertung und Vergleich der Ansätze

4.1 Anhand des Kriterienkatalogs

4.1.1 Was wird unter einem Fragment verstanden und wie wird es definiert?

A Keine explizite Definition. Verstanden wird darunter eine zusammenhängende Menge von Aktivitäten, welche bei einem beliebigen Partner, als Teil eines Workflows ausgeführt werden kann.

B Ein Fragment ist der Teil des Prozessmodells der auf einer bestimmten Workflow-Maschine ausgeführt wird.

C Ein Fragment ist ein Stratum, das aus einer Menge von *Basic Transactions* besteht. Eine *Basic Transaction* ist ein atomarer Task.

I Keine explizite Definition. Angenommen wird, dass ein Fragment eine Menge von BPEL-Aktivitäten ist.

4.1.2 Was ist die Granularität der Fragmentierung?

A Aktivitäten können beliebig auf Fragmente verteilt werden.

B Aktivitäten können beliebig auf die Teilnehmer (physischen Maschinen) auf denen das Modell ausgeführt wird verteilt werden.

C Atomare Tasks (Aktivität)

I Die Granularität ist auf Aktivitäts-Ebene, da einzelne Aktivitäten von Fragment zu Fragment verschoben werden können. Eigentlich geht es aber um die einzelnen Webservices, die jeweils einem eigenen Fragment zugeteilt sind und deren Glue-Code zwischen den Fragmenten hin und her geschoben werden kann.

4.1.3 Wieso wird fragmentiert? Was sind die Ziele und Motivationen des Ansatzes?

A Ermöglicht es die einzelnen Fragmente dezentral, von verschiedenen Partnern, auch über Firmengrenzen hinweg ausführen zu können. Das Verfahren kann beispielsweise beim Aufbau firmenübergreifender Prozesse und beim Outsourcing von bestehenden Prozessen verwendet werden. Ziel ist die Kostenreduzierung.

B Fragmentiert wird, damit BPEL-Prozesse dezentralisiert und verteilt ausgeführt werden können. Besonderes Augenmerk wird dabei darauf gelegt, dass die Prozessmodelle dem Geschäftsziel entsprechen und nicht der verwendeten Infrastruktur angepasst werden.

C - Verteilung der Ausführung auf mehrere Maschinen,
- Out-/In-Sourcing,
- Optimieren von Prozessen in Bezug auf Nebenläufigkeit und Antwortzeiten.

I - Ziel ist es den Durchsatz des Prozessmodells zu erhöhen, indem Teile des Glue-Codes auf die Knoten mit den Webservices verschoben werden und die Fragmente direkt untereinander kommunizieren.
- Erhöhte Parallelität
- Minimierung Kommunikationskosten

4.1.4 Nach welchen Kriterien werden die Aktivitäten auf die Fragmente verteilt?

A - Der Prozessmodellierer ordnet die Aktivitäten von Hand den Fragmenten zu, je nach Business-Anforderungen, ohne technische Einschränkungen.
- Die Aufteilung der Fragmente könnte auch automatisiert ablaufen, wenn jede Aktivität "weiß" in welcher Organisation, oder auf welchem Server (nahe bei den benötigten Ressourcen), sie ablaufen soll.
- Es wird keine Aussage darüber getroffen wie die Identifikation der Fragmente abläuft. Wie oben erwähnt sind aber Organisationsgrenzen oder Ressourcen gute Kandidaten.

B - Die Aktivitäten werden den Maschinen zugewiesen, die möglichst nahe an den auszuführenden Services liegen, um z.B. das Transferieren von großen Datenmengen zu verhindern.
- Die Fragmentierung kann manuell oder automatisiert (basierend auf festgelegten Parametern oder einer Kostenfunktion) erfolgen. Über die genauen Details der automatisierten Fragmentierung werden keine Angaben gemacht.

C Die Fragmente werden nach bestimmten Kriterien wie z.B. Antwortzeit, Ausführungszeit, Nachrichtenkosten, etc. fragmentiert.

I Optimierung des Durchsatzes und Minimierung der übertragenen Daten.

4.1.5 Wird die Fragmentierung zur Design- oder Laufzeit durchgeführt?

A Designzeit

B Designzeit

C Momentan zur Design-Zeit, zukünftige Arbeiten sollen es zur Laufzeit ermöglichen.

I Designzeit

4.1.6 Wie werden BPEL-Konstrukte unterstützt?

A - Einige BPEL-1.0-Konstrukte sind grundsätzlich verboten: terminate, throw, compensate und ein assign welches in einen Endpunkt kopiert, request/reply muss im gleichen Fragment sein. ([15], 5.2)
- Fault- und/oder Compensation-Handler werden unterstützt, Eventhandler nicht. Dead-Path-Elimination muss immer aktiviert sein, wird also unterstützt.
- Correlation wird insofern unterstützt, dass es ein sogenanntes globales Correlation Set pro Prozessmodell gibt, das dann über alle Fragmente hinweg in der Kommunikation der Fragmente verwendet wird.

B - Alle BPEL-Konstrukte werden unterstützt. Dadurch, dass der Originale BPEL-Prozess vorgehalten wird, kann auch jederzeit auf die entsprechenden Daten zugegriffen werden.
Bei DPE werden Dead-Path-Tuple von Aktivität zu Aktivität weitergegeben.
- Compensation und Exception Handling wird im Petri-Netz-Semantik behandelt

C - Compensation, Exception Handling und Correlation wie bei BPEL
- Durch einen vereinfachten Kontrollfluss wird keine DPE unterstützt

I Unterstützt wird ein, im Paper nicht exakt spezifiziertes Subset von BPEL, welches folgende BPEL-Konstrukte und teilweise zusätzliche Erweiterungen enthält:
flow, sequence, switch, while, pick, flow, link, variable, invoke (synchroner, blockierender Aufruf), send (asynchroner, unidirektionaler, nicht-blockierender Aufruf, wäre in BPEL auch invoke), receive, reply, assign und compute (arithmetische oder logische Operation).

4.1.7 Was für ein Prozessmodell wird als Eingabe verwendet?

A BPEL-D wird als Eingabe verwendet. Es kann aus einem vorhandenen BPEL-Modell generiert werden, worauf aber nicht eingegangen wird.

B BPEL

C Ein beliebiges Prozessmodell, das in Form von Transaktionen dargestellt wird.

I BPEL

4.1.8 Welche zusätzlichen Änderungen am Prozessmodell sind vor der Fragmentierung notwendig?

A Das Prozessmodell muss als BPEL-D vorliegen, aber im Voraus nicht anderweitig verändert werden.
Beim Erstellen der BPEL Prozesse für die einzelnen Fragmente, wird aber eine größere Anzahl von weiteren BPEL-Aktivitäten und Scopes hinzugefügt.

B Keine davor, bei der Fragmentierung zusätzliche Information darüber welche Aktivitäten und Variablen auf die Teilnehmer verteilt werden (Deployment Descriptor), dann Umwandlung in Petri-Netz.

C Keine, allerdings muss bekannt sein, welche Ressourcen verwendet werden.

I Vom Benutzer sind keine Änderungen nötig. Am Prozessmodell ebenfalls nicht, es wird lediglich in einen Datenabhängigkeits-Graphen umgewandelt der zur Ermittlung der optimalen Fragmentierung verwendet wird.

4.1.9 Können vorhandene Standard-Modellierungs-Tools (beispielsweise für BPEL oder BPMN) weiter genutzt werden?

A Standard-Modellierungs-Tools für BPEL können weiterverwendet werden. Die Zuordnung der Aktivitäten zu Fragmenten, sowie die Umwandlung in BPEL-D werden von diesen nicht unterstützt und müssen, nach der Fragmentierung, außerhalb des Tools gelöst werden.

B Alle existierenden Tools können weiter genutzt werden

C Ja, allerdings muss beachtet werden, dass der Kontrollfluss nur Split und And-Joins unterstützt.

I Ja, auf die resultierende BPEL Datei wird dann das beschriebene Verfahren angewendet.

4.1.10 Können bestehende, bereits modellierte Prozessmodelle fragmentiert werden?

A Ja, da das Verfahren sowieso nach der Modellierung des Prozessmodells angewendet wird, kann es auch auf bestehende Prozessmodelle angewendet werden.

B Ja, ohne Probleme. Die Autoren legen explizit Wert darauf, dass das bisherige Vorgehen bei der Modellierung nicht verändert werden muss.

C Ja, wenn das Prozessmodell den bereits genannten Einschränkungen genügt.

I Ja, da das Verfahren erst nach Abschluss der Modellierung angewendet wird.

4.1.11 Ist die Wiederverwendung der Fragmente vorgesehen?

A Die Wiederverwendung von Fragmenten in mehreren Prozessmodellen ist nicht vorgesehen (wird im Ansatz nicht betrachtet) und wird aufgrund der Einschränkungen von BPEL-D zusätzlich erschwert.

B Die Wiederverwendung ist nicht explizit vorgesehen. Grundsätzlich sollte sie aber möglich sein.

C Ja

I Nicht explizit vorgesehen. Die Fragmente, mit „eingebetteten“ Webservice, könnten aber in anderen Prozessen eingebunden werden. Diese müssten dann, falls der Glue-Code um den Webservice herum die Daten weiterverarbeitet, aber auch entsprechende Daten senden. Falls gar kein oder nur sehr wenig Glue-Code in dem Fragment ist, ist es aber sicher besser einfach den Webservice direkt zu verwenden.

4.1.12 Ist zur Ausführung des fragmentierten Prozesses eine normale BPEL-Engine ausreichend, oder werden zusätzliche Komponenten oder Erweiterungen benötigt?

A Eine standardkonforme BPEL-Engine ist ausreichend.

B Es wird ein Workflow-System benötigt, das EWFN ausführen kann. Ebenfalls benötigt wird die Implementierung eines Tuplespaces zur Kommunikation.

C Eine standardkonforme BPEL-Engine ist ausreichend.

I Eine standardkonforme BPEL-Engine ist ausreichend.

4.1.13 Wie findet die Kommunikation zwischen den Fragmenten statt?

A Die Fragmente rufen sich über die in BPEL verfügbaren Mechanismen (*Invoke*-Aktivität) auf, nutzen also Webservices auf Basis einer WSDL.

B Die Kommunikation zwischen den einzelnen Aktivitäten und dadurch auch zwischen den Fragmenten findet über Tuplespaces statt. Für die Kommunikation zwischen Fragmenten wird ein globaler Tuplespace verwendet, innerhalb der Fragmente werden lokale Tuplespaces verwendet.

C Die Kommunikation findet über persistente Message-Queues statt

I Messaging

4.1.14 Wie groß ist der Overhead der Fragmentierung?

A Gerade bei kleinen Fragmenten kann der Overhead verhältnismäßig hoch sein, da (je nach dem was übertragen wird, Kontroll- und/oder Datenfluss) mehrere zusätzliche Aktivitäten und Scopes, sowie die dazugehörigen Handler in den BPEL-Prozess hinzugefügt werden müssen.
Durch die Einführung des expliziten Datenflusses werden nur die benötigten Daten übertragen (mit dem Overhead den ein WSDL Webservice mit sich bringt). Es werden die BPEL Variables also weder global zugänglich gemacht, noch alle von Fragment zu Fragment übertragen, ohne das deren Inhalt benötigt wird.

B Keine genauen Angaben.

C Der Overhead ist unter anderem abhängig von den verwendeten Ressourcen und der Anzahl an Nachrichten die zwischen den einzelnen Strata versendet werden müssen, sowie den jeweiligen, unter 2.2.3 genannten Eigenschaften der *Basic Transactions*.

I - Kein großer Overhead, da die Webservices ja sowieso aufgerufen werden müssen. Es werden in vielen Fällen sogar weniger Daten gesendet, da diese direkt an die nächste Aktivität gesendet werden, anstatt zurück zum eigentlichen Prozess.
- Sei es durch die Kommunikation oder durch das Hinzufügen von Hilfs-Konstrukten (höhere Anzahl der Elemente die navigiert werden müssen).

4.1.15 Wird der Workflow von einem zentralen Koordinator kontrolliert? Kann der globale Zustand des Workflows eingesehen werden, oder ist dieser über die Fragmente verteilt?

A Es gibt keinen zentralen Koordinator und der Zustand einer Instanz eines Workflows kann nicht direkt eingesehen, sondern es müssten alle Fragmente konsultiert werden.
Je nach Aufteilung der Fragmentierung kann aber das „Start-Fragment“ gewisse Rückschlüsse über den Fortschritt liefern. Beispielsweise wenn eine while-Schleife so fragmentiert wird, dass die Schleifen-Steuerung im Start-Fragment ist und der Inhalt in einem eigenen Fragment.

B Es gibt keinen zentralen Koordinator, die Aktivitäten koordinieren sich selber. Dadurch und durch grundlegende Eigenschaften von Petri-Netzen ist es auch nicht möglich den globalen Zustand einzusehen.

C Der globale Zustand kann nicht eingesehen werden, er ist über die Fragmente verteilt.

I - Es gibt keinen zentralen Koordinator, die Fragmente übertragen die Daten selbständig an die Fragmente welche diese benötigen.
- Der globale Zustand kann nicht eingesehen werden, da er über die Fragmente verteilt ist

4.1.16 Wird nur die verteilte Ausführung von Fragmenten beschrieben, oder auch wie eine optimale Fragmentierung gefunden wird?

A Hilfestellung zum Ermitteln einer optimalen Fragmentierung gibt es nicht.
In diesem Ansatz steht aber nicht die Effizienz der Fragmentierung im Mittelpunkt, sondern Business-Szenarien wie die Ausführung eines firmenübergreifenden Prozesses. Dort müssen die Aktivitäten beispielsweise so aufgeteilt werden, dass die Aktivitäten, welche interne Daten einer Firma benötigen, auch in dieser ausgeführt werden und so Firmengeheimnisse gewahrt bleiben.

B Es wird nur die verteile Ausführung beschrieben, zur Findung der optimalen Fragmentierung wird nur die Aussage gemacht, dass diese mittels Optimierungsalgorithmen, wie lokale Suche oder Hill-Climbing gefunden werden könnte.

C Es werden Optimierungsalgorithmen vorgeschlagen, insbesondere ein neuer Ansatz, der Hill Climbing und Simulated Annealing kombiniert.

I Es wird sehr ausführlich auf das Finden einer für den Durchsatz optimalen Fragmentierung eingegangen.

4.2 Tabellarischer Vergleich ausgewählter Gesichtspunkte

Um bei einem vorliegenden Prozessmodell einfacher entscheiden zu können welches Verfahren eingesetzt werden könnte, haben wir Übersichts-Tabelle für ausgewählte Gesichtspunkte entwickelt.

4.2.1 In welchem Format muss das Prozessmodell vorliegen?

Paper	Keine Angabe	BPEL	Petri-Netz	Andere Notation
A		(X)		BPEL-D
B		X		
C				Prozessmodell, das aus Transaktionen besteht
D		X		
E				SESE
F				Eigenes, aufbauend auf Information Control Net
G	X			
I		X		
J			X	
K			X	WF-net
L				Grafischer Workflow Graph

Tabelle 1: In welchem Format muss das Prozessmodell beim jeweiligen Ansatz vorliegen

5 Szenario: Anwendung der Ansätze auf ein Prozessmodell

Um die Anwendung des Ansatzes an einem realen Prozessmodell und nicht nur auf dem Papier sehen zu können, haben wir ein einfaches Szenario entwickelt, auf welches die verschiedenen Verfahren angewendet werden. Dazu haben wir das allgemein bekannte "*LoanApproval*"-Prozessmodell aus der BPEL-2.0-Spezifikation [16] gewählt, welches wir um eine zusätzliche Überprüfung der Adresse erweitert haben (Aktivitäten: "*InvokeAddressCheck*" und "*AssignNo*").

Der Quellcode der BPEL- und WSDL-Dateien sind in Kapitel 10 zu finden.

5.1 Das Szenario: "LoanApprovalExtended"

Die "*ReceiveRequest*"-Aktivität erhält eine Kreditanfrage, welche Vor-, Nachname, Adresse und den gewünschten Betrag enthält. Falls der Betrag >= 10000 ist, wird der Link zur Adressüberprüfung navigiert, wenn nicht wird das Risiko von einem externen Webservice ermittelt.
Falls das Risiko gering ist, wird dem Antrag zugestimmt und das Ergebnis in "*ReplyResult*" gesendet, wenn nicht wird auch eine Adressüberprüfung durchgeführt.
Die Adressüberprüfung wird von einem externen Webservice durchgeführt (der z.B. von einem Dienstleister mit einer umfangreichen Adressdatenbank betrieben wird). Ist die Adresse falsch, wird der Antrag abgelehnt und das Ergebnis gesendet.
Wenn die Adresse korrekt ist, wird der "*LoanApproval*" Webservice aufgerufen, welcher den Kreditantrag prüft (z.B. durch einen Bankangestellter) und annimmt oder ablehnt. In beiden Fällen wird die Antwort des Angestellten dann direkt zurückgesendet (deshalb ist auf diesem Link kein zusätzliches "*Assign*" nötig).

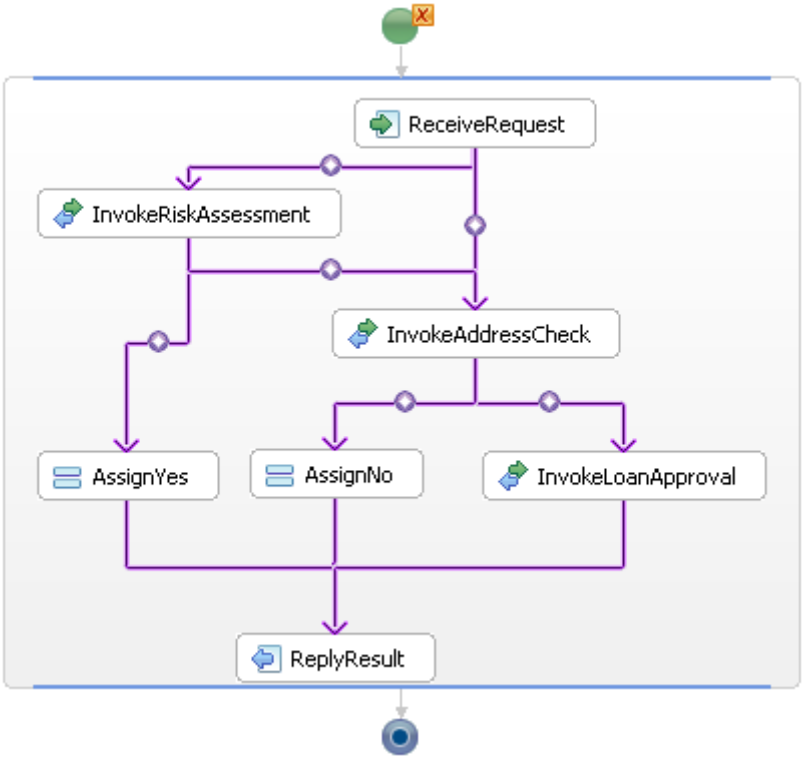

Abbildung 3: Prozessmodell "LoanApprovalExtended"

5.2 A: Role-based Decomposition of Business Processes using BPEL

5.2.1 Fragmentierung

Der Prozessmodellierer hat sich für folgende Fragmentierung entschieden, wobei alle Fragmente außer dem *Address*-Fragment lokal im Unternehmen ablaufen, nur der *AdressCheck* wurde ausgelagert an eine externe Firma.

Diese Fragmentierung ist natürlich nur bedingt sinnvoll, da *invoke*-Aktivitäten wie *InvokeLoanApproval* in einem eigenen Fragment keinen Sinn machen, da man das *Invoke* dann auch im Prozess belassen könnte, weil ja der Web-Service-Aufruf auch eine Art der Fragmentierung ist.

Da wir die Fragmentierung rein grafisch durchführen, müssen wir keine Umwandlung von BPEL in BPEL-D durchführen, wobei diese auch nicht nötig wäre, da im Prozessmodell keine verbotenen Konstrukte (siehe: 2.2.1) verwendet werden.

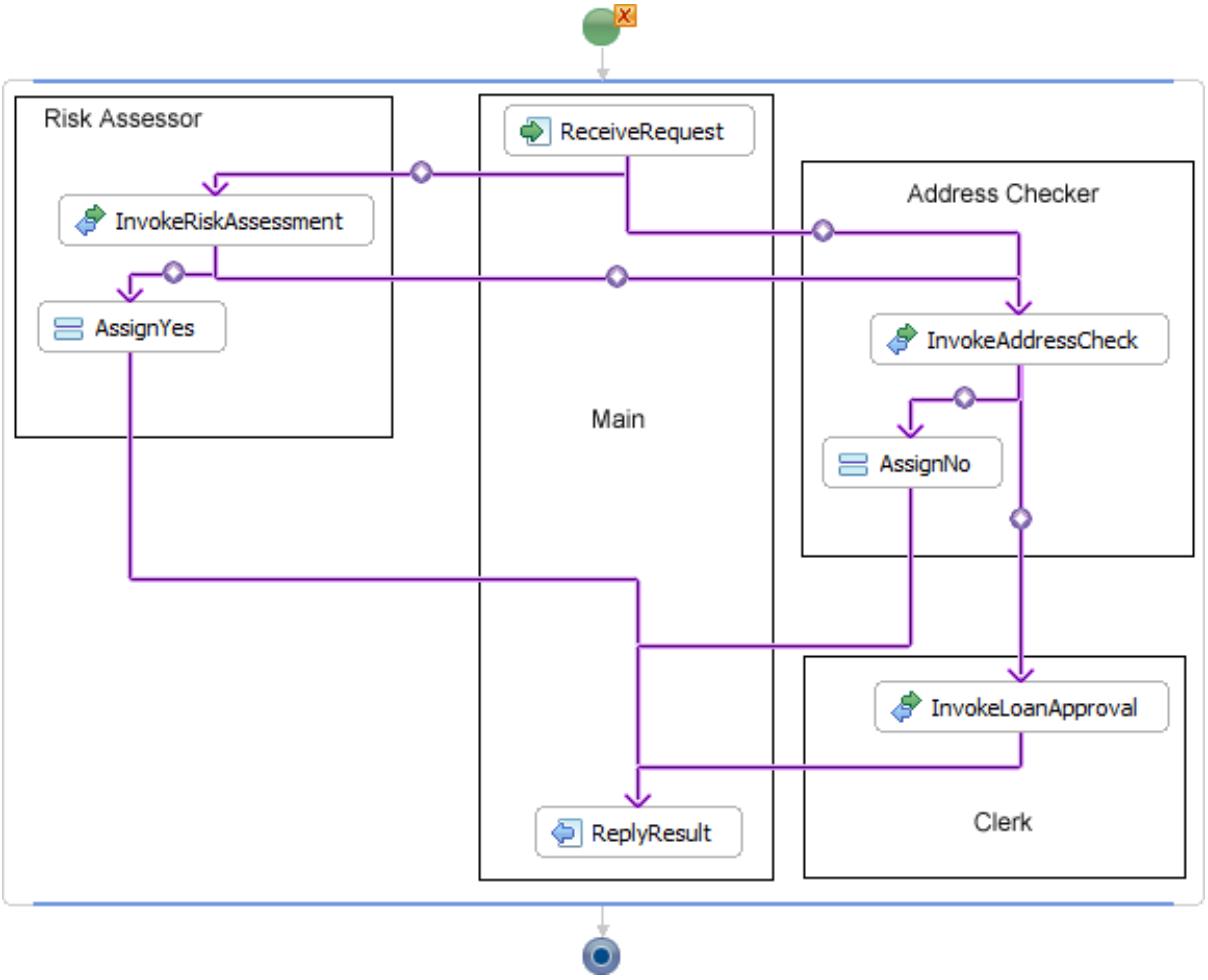

Abbildung 4: Fragmentierung des Prozessmodells

5.2.2 Aufgebrochene Links

Nach der Fragmentierung gibt es 7 aufgebrochene Verbindungen, die folgende Eigenschaften (und Nummerierungen) haben:
Mit jedem aufgebrochen Link wird neben dem Kontrollfluss, auch Daten explizit über den Link übertragen. Diese werden unten jeweils angegeben, wobei es in diesem Beispiel immer nur exakt eine komplette BPEL Variable ist.

1. **Link:** ReceiveRequest -> InvokeRiskAssesment
   Fragment: Main -> Risk
   Data Variable: request:creditInformationMessage
2. **Link:** InvokeRiskAssesment -> InvokeAddressCheck
   Fragment: Risk -> Address
   Data Variable: request:creditInformationMessage
3. **Link:** AssignYes -> ReplyResult
   Fragment: Risk -> Main
   Data Variable: approval:approvalMessage
4. **Link:** ReceiveRequest -> InvokeAddressCheck
   Fragment: Main -> Address
   Data Variable: request:creditInformationMessage
5. **Link:** AssignNo -> ReplyResult
   Fragment: Address -> Main
   Data Variable: approval:approvalMessage
6. **Link:** InvokeAddressCheck -> InvokeLoanApproval
   Fragment: Address -> Clerk
   Data Variable: request:creditInformationMessage
7. **Link:** InvokeLoanApproval -> ReplyResult
   Fragment: Clerk -> Main
   Data Variable: approval:approvalMessage

5.2.3 Aufgebrochene Links wiederherstellen

Zur Wiederherstellung müssen in die Fragmente, in denen Anfang bzw. Ende des Links liegt, jeweils neue BPEL-Konstrukte eingefügt werden, welche den Kontroll- bzw. Datenfluss entsprechen in das anderen Fragment übertragen. Wird das gemacht, so verhält sich der fragmentierte Prozess genauso wie davor, als nicht fragmentierter Prozess. Im Beispiel sind alle aufgebrochen Links praktisch gleich, deshalb beschreiben wir die Wiederherstellung des Links nur allgemein.
Für einen Link von A nach B, sehen die BPEL-Konstrukte für den wiederhergestellten Link dann wie folgt aus:

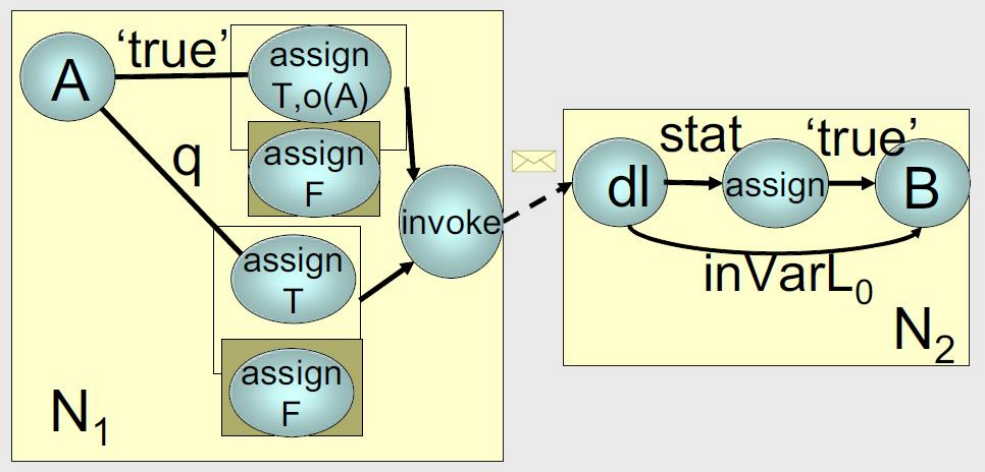

Abbildung 5: Wiederhergestellter Link (Quelle: [1], Figure 8)

Für die Übertragung des Kontrollflusses sind in N1 die unteren beiden *assigns* zuständig. Vor deren Ausführung wird die bisherige Transition Condition q evaluiert. Für die Übertragung des Datenflusses, in den oberen beiden *assigns*, ist die Transition Condition *true*, da wir die Daten unabhängig von q (und dem Kontrollfluss) übertragen müssen.
Das eigentliche *assign* ist jeweils in einem Scope, mit *suppressJoinFailure=false*, damit falls q *false* ist der FaultHandler mit einem Join Failure aufgerufen wird (sonst könnte das Ergebnis von q gar nicht ermittelt werden). Das *invoke* fasst die beiden Zweige zusammen und sendet die Daten an das andere Fragment.
Im zweiten Fragment N2 wird Daten- und Kontrollfluss von der *receive*-Aktivität dl empfangen. Im *assign* werden die Daten von A in die Input Variable von B kopiert, entsprechend der früheren Data Map des Links A-B.

Nachdem an allen Fragmentgrenzen die entsprechenden Konstrukte eingeführt wurden, gibt es für jedes Fragment eine BPEL. Außerdem bekommt jedes Fragment eine eigene WSDL Datei, welche nur die PartnerLinks enthält, welche im Fragment benötigt werden. Also im "Risk"-Fragment wird nur der PartnerLink für InvokeRiskAssesment benötigt. Außerdem werden die nötigen PartnerLinks für die Kommunikation mit den anderen Fragmenten hinzugefügt.

5.3 B: A Novel Approach to Decentralized Workflow Enactment

Das Verfahren wird in 4 Schritten durchgeführt:

1. Process Modelling
2. Process Segmentation
3. Process Deployment
4. Distributed Process Execution, wird in diesem Beispiel nicht betrachtet

Im Folgenden wird das Test-Szenario mit dem Fragmentierungs-Ansatz B durchgeführt.

5.3.1 Schritt 1

Schritt 1, also die Prozessmodellierung wurde bereits in Kapitel Das Szenario: "LoanApprovalExtended" durchgeführt und erläutert.

5.3.2 Schritt 2

In Schritt 2 wird das Prozessmodell auf die physische Infrastruktur, die den Prozess später ausführen soll, verteilt.

Dazu wird angenommen, dass folgende Workflow-Maschinen im Kreditinstitut existieren:

- **Haupt-Maschine:** Verantwortlich für das Starten und Beenden der Prozess-Instanzen.
- **Risiko-Berechnung**: Die Maschine führt die Berechnung des Risikos für Kreditanträge durch.
- **Adress-Maschine**: Ist für die Validierung von Adressdaten zuständig.
- **Sachbearbeiter-Maschine:** Diese Maschine ist dafür verantwortlich die Kreditanträge, die von Sachbearbeitern geprüft und genehmigt/abgelehnt werden müssen zu verwalten.

Die jeweils nötigen Webservices für die einzelnen Aktivitäten liegen auf den entsprechenden Maschinen. Die manuelle Aufteilung des Prozesses auf die 4 Maschinen ist in Abbildung 6 dargestellt:

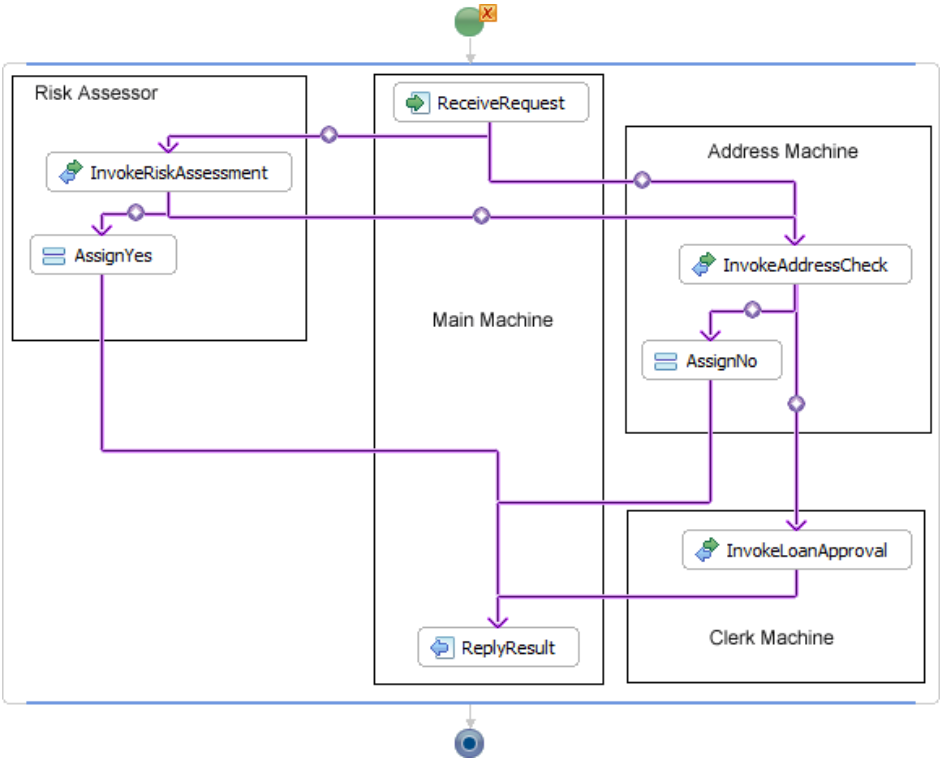

Abbildung 6: Grafische Darstellung der Fragmentierung

Wie leicht zu erkennen ist, wurde jede Aktivität der Maschine zugewiesen, die den benötigten Webservice bereitstellt.

Dadurch wird, wie vom Ansatz vorgesehen, eine effizientere Ausführung des Prozesses möglich gemacht.

5.3.3 Schritt 3

In Schritt 3 wird das BPEL-Modell in ein EWFN-Modell umgewandelt.

Zur übersichtlichen Darstellung sind alle EWFN-Pattern in diesem Beispiel leicht vereinfacht dargestellt. Es wurde auf die Places für die Prozess-Metadaten, also das Originale Prozess-Modell, das als XML vorgehalten wird verzichtet. Ebenfalls weggelassen wurden die Places für Fehler.

Im Folgenden werden nun die 4 Fragmente in EWFN-Notation dargestellt:

In Abbildung 7 zu sehen ist das Hauptfragment mit den beiden EWFN-Pattern für die *Receive*- und *Reply*-Aktivität des Prozesses.a

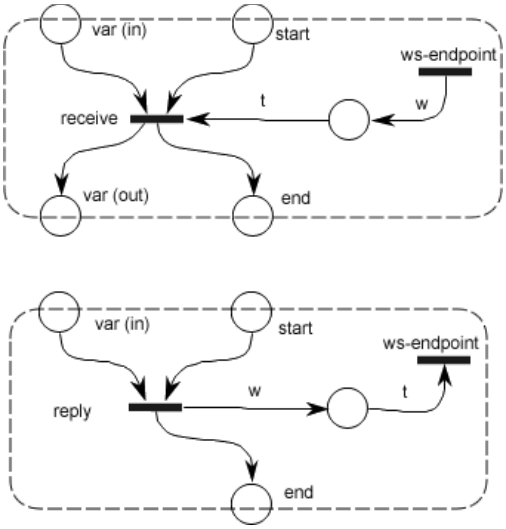

Abbildung 7: Hauptfragment

Abbildung 8 stellt das *Risk Assessor*-Fragment dar, das im Aufbau exakt mit dem *Address Checker*-Fragment übereinstimmt.

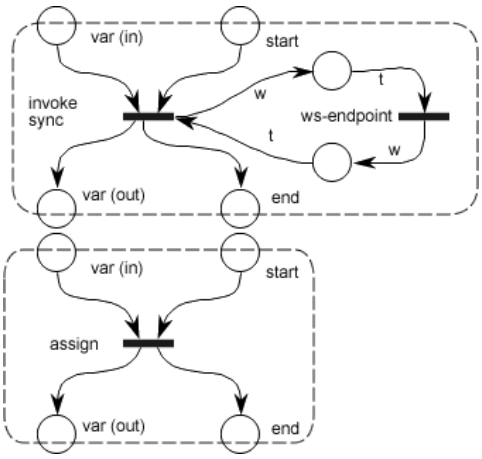

Abbildung 8: Risk Assessor/Address Checker

In Abbildung 9 ist der asynchrone Aufruf des *LoanApproval*-Webservices zu sehen.

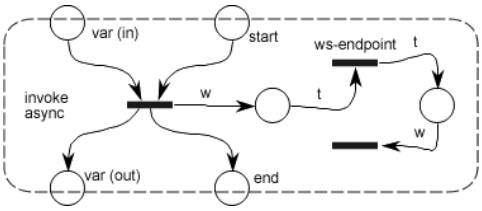

Abbildung 9: Loan Approval

5.4 C: Optimal Stratification of Transactions

Das in Kapitel 2.2.3 vorgestellte Prozessmodell wird in eine Transaktionen-Darstellung umgewandelt. Jede Aktivität wird dabei als eine *Basic Transaction* dargestellt.

Im Fall des Beispiels entsteht dann ein Graph, der folgende Transaktion beinhaltet:

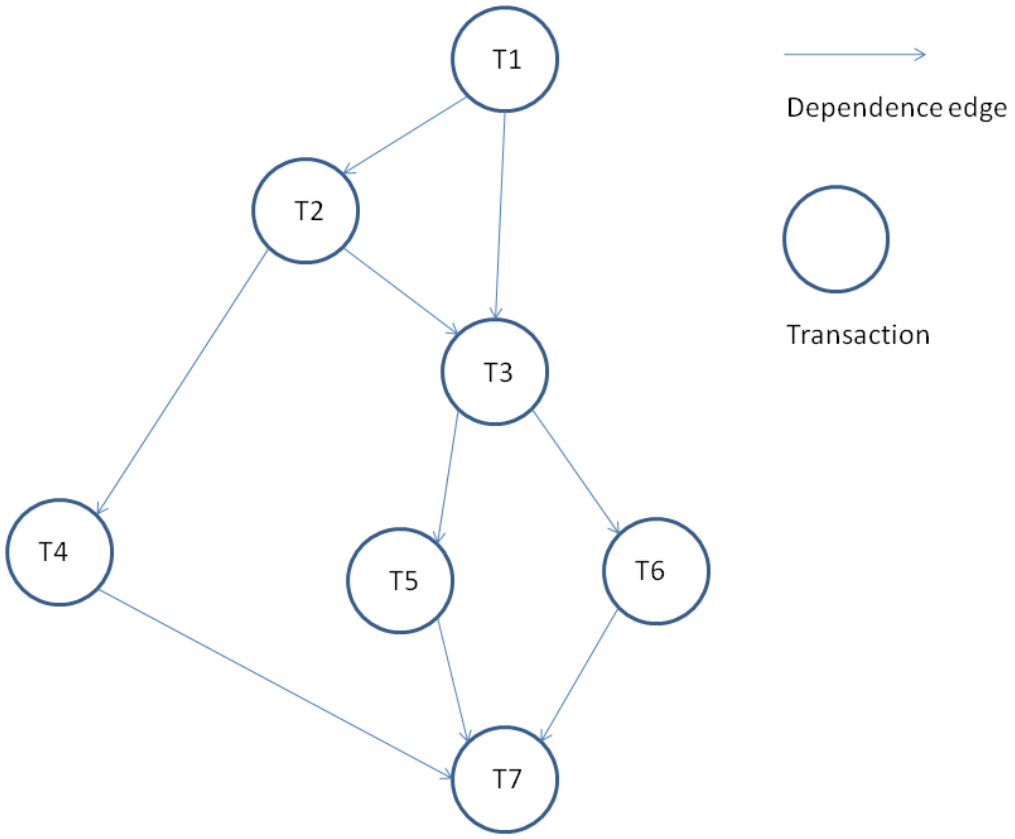

Abbildung 10: Dependence Graph

Wobei die Transaktionen folgenden Aktivitäten des Prozessmodells entsprechen:

- T1: ReceiveRequest
- T2: InvokeRiskAssessment
- T3: InvokeAddressCheck
- T4: AssignYes
- T5: AssignNo
- T6: InvokeLoanApproval
- T7: ReplyResult

Anhand verschiedener Kriterien wie z.B.:

- Kosten, die sie verursachen

- Abbruchwahrscheinlichkeit
- Ausführungsdauer
- Startzeit

werden die Transaktionen mit Hilfe von Graphoptimierungs-Algorithmen optimal in sogenannte Strata zusammengefasst. Wir nehmen an, dass die optimale Verteilung des LoanApproval-Extended-Prozessmodells folgende, in Abbildung 11 zu sehende Verteilung wäre.

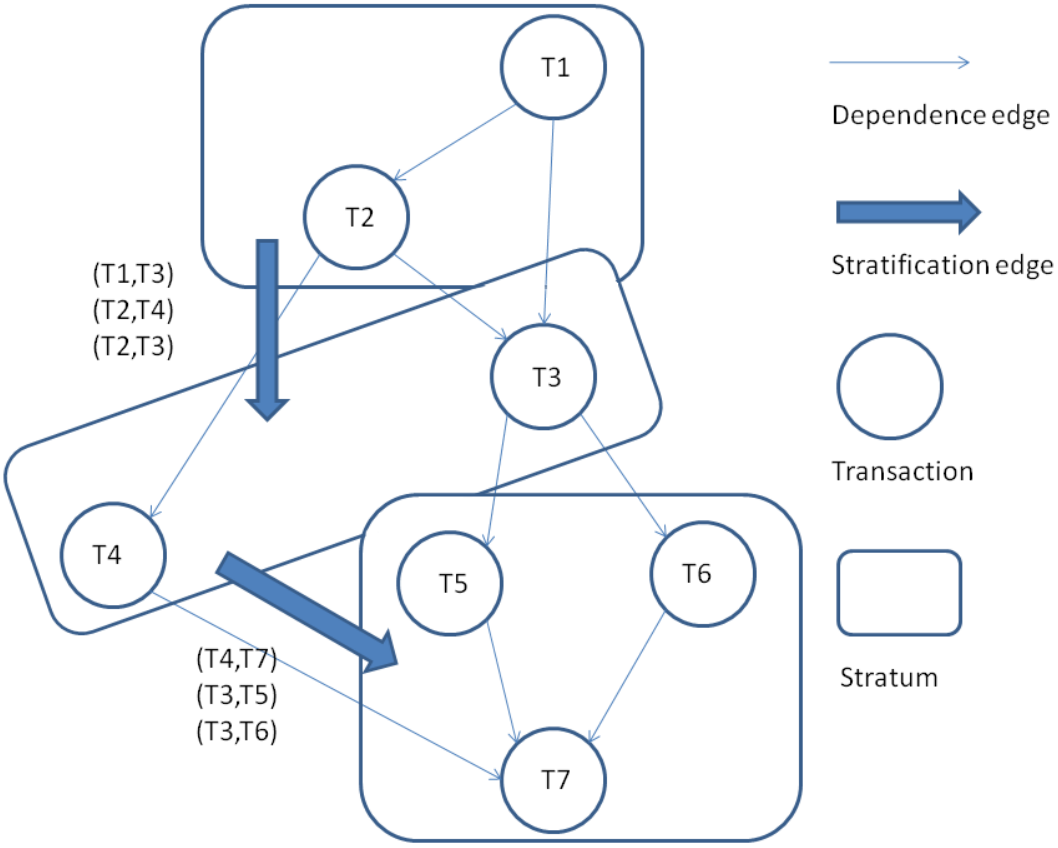

Abbildung 11: Stratification Graph

5.4.1 Formale Darstellung der Graphen

Stratification graph:

$G_s = (V_s, E_s)$

$V_s = \{S1, S2, S3\}$
$E_s = \{(S1, S2), (S2,S3)\}$

edgeSet((S1, S2)) = {(T1, T3), (T2, T4)}
edgeSet((S2, S3)) = {(T4, T7), (T3, T6)}

Dependence graph

$G_d = (V_d, E_d)$

$V_d = \{T1, T2, T3, T4, T5, T6, T7\}$
$E_d = \{(T1, T2), (T1, T3), (T2, T3), (T2, T4), (T3, T5), (T3, T6), (T4, T7), (T5, T7), (T6, T7)\}$

5.5 I: Decentralizing execution of composite Webservices

5.5.1 Threaded Control Flow Graph (TCFG)

Aus dem BPEL-Code (siehe Anhang) lässt sich leicht ein TCFG ableiten, der starke Ähnlichkeit zur normalen BPEL-Modellierung des Prozessmodells hat.
In diesem Graph sieht man sehr gut welche Daten bei den Invokes benötigt werden und welche "raus kommen", was für die Umwandlung in einen Programmabhängigkeits-Graphen im nächsten Schritt sehr wichtig ist.
Hervorzuheben ist die Verbindung von p2 nach F2, welche sozusagen die IF-Strukturen des Graphen sprengt. Dies kann in BPEL, mit Links, ohne Probleme abgebildet werden, hier ist dafür eine sogenannte *Synchronization Dependency* (gestrichelte Linie) nötig.
Die anderen Pfeile bilden den normalen Kontrollfluss ab.

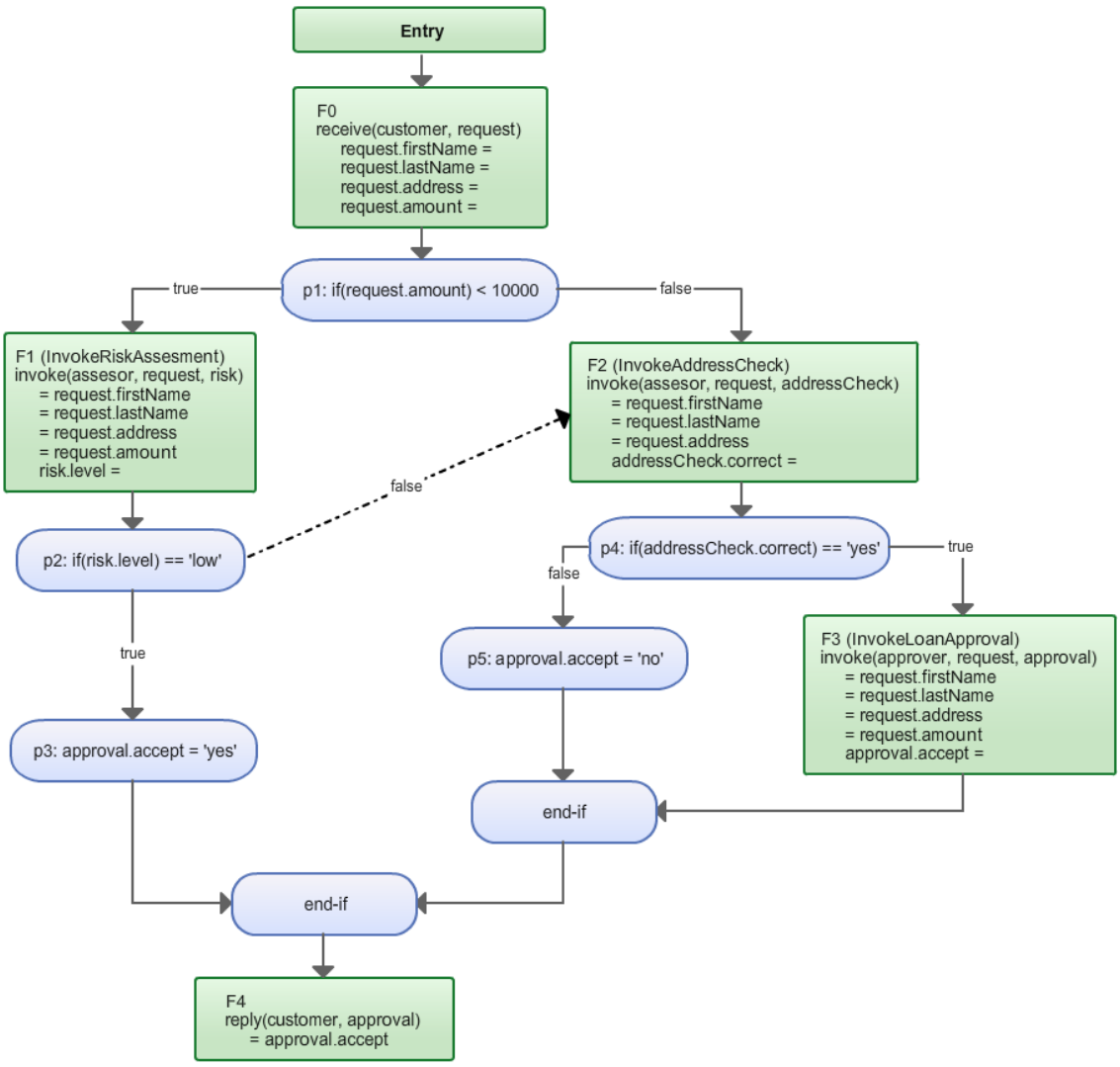

Abbildung 12: Threaded Control Flow Graph

5.5.2 Program Dependence Graph (PDG)

Aus dem TCFG wird jetzt ein *Program Dependence Graph* gebildet, wie im Paper beschrieben.
Hierzu werden hauptsächlich Datenabhängigkeiten eingefügt, beispielsweise benötigt F4 (*reply*) die *approval*-Variable von F0, p3, p5 und F3, weil es diese ja entsprechend zurückgeben muss.

Die grünen Kästen sind feste Aktivitäten (diese müssen auf diesem Knoten ausgeführt werden, weil dort beispielsweise die benötigten Daten vorliegen), blaue abgerundete Kästen sind portable Aktivitäten, die in verschiedenen Fragmenten sein könnten.
Die blauen dünnen Linien sind Datenabhängigkeiten, die Anderen haben die gleiche Bedeutung wie beim TCFG.

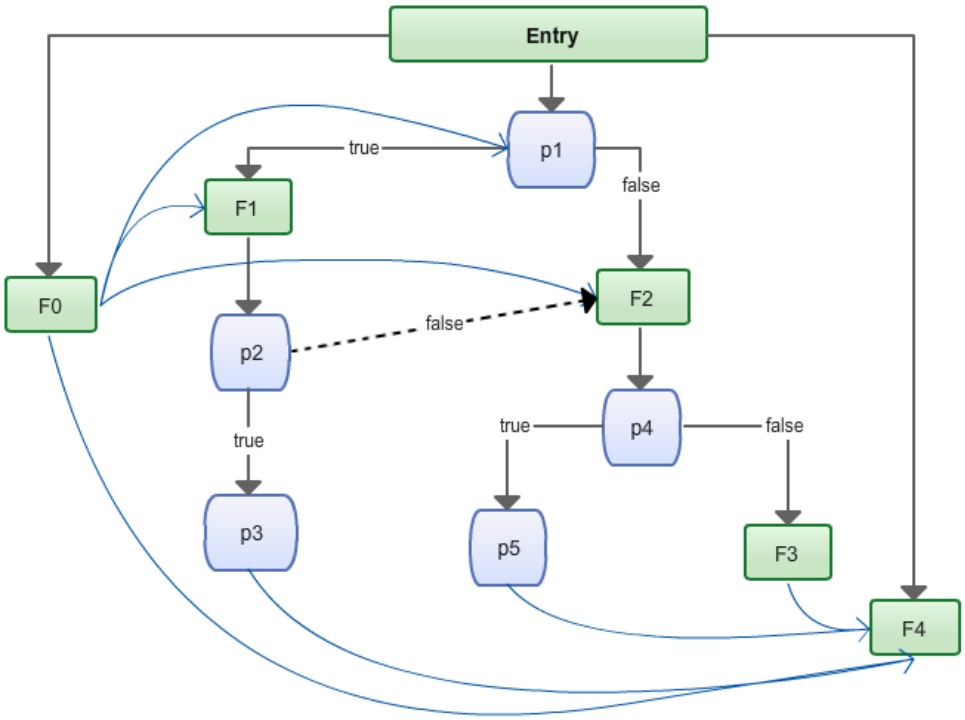

Abbildung 13: Program Dependence Graph

5.5.3 Partitionierung

In der eigentlichen Fragmentierung wird jetzt mit dem PDG weiter gearbeitet: Zuerst sehen wir das p3 kontrollabhängig von p2 ist, keine Kinder hat und portabel ist. Deshalb wird p2 mit p3 zusammengelegt und als portabel markiert, da beide Teile portabel waren.

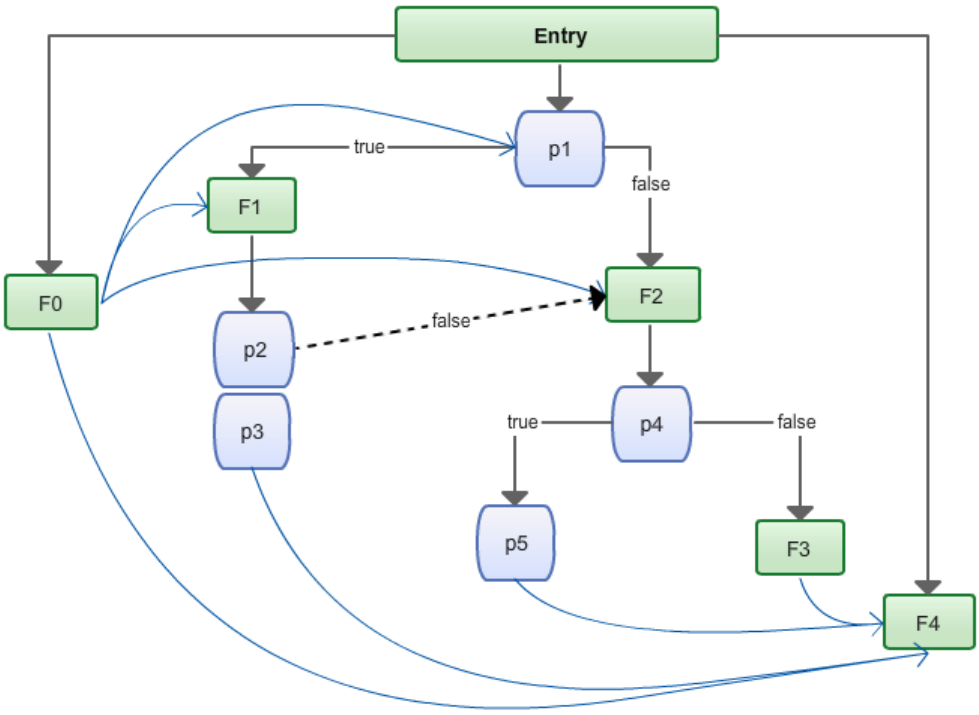

Abbildung 14: Program Dependence Graph, Ausgangszustand

Als nächstes sieht man, dass das Fragment {p2, p3} keine Kinder hat, portabel ist und kontrollabhängig von F1 ist, weshalb alle drei in ein neues Fragment zusammengelegt werden, welches nicht mehr portabel, sondern fest bei F1 ist.

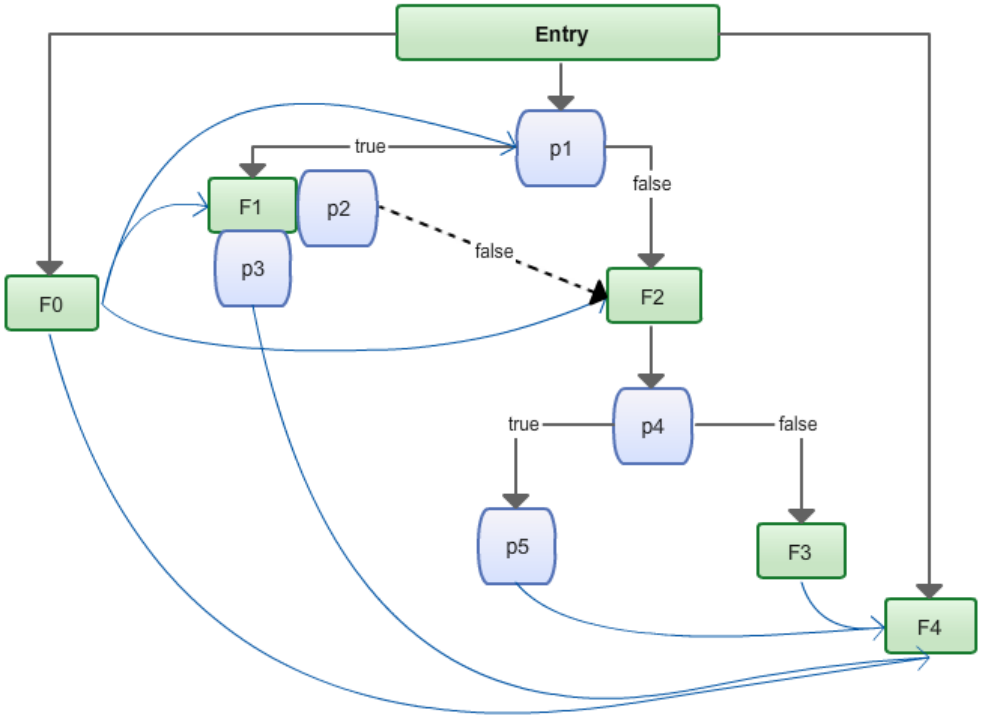

Abbildung 15: Program Dependence Graph, Schritt 1

Nun können wir p5 und F3 bearbeiten, die den *true*- und *false*-Zweig der IFs in p4 bilden. Beide sind kontrollabhängig von p4 und haben jeweils keine Kinder. Deshalb werden alle drei in ein nicht portables Fragment zusammengelegt.

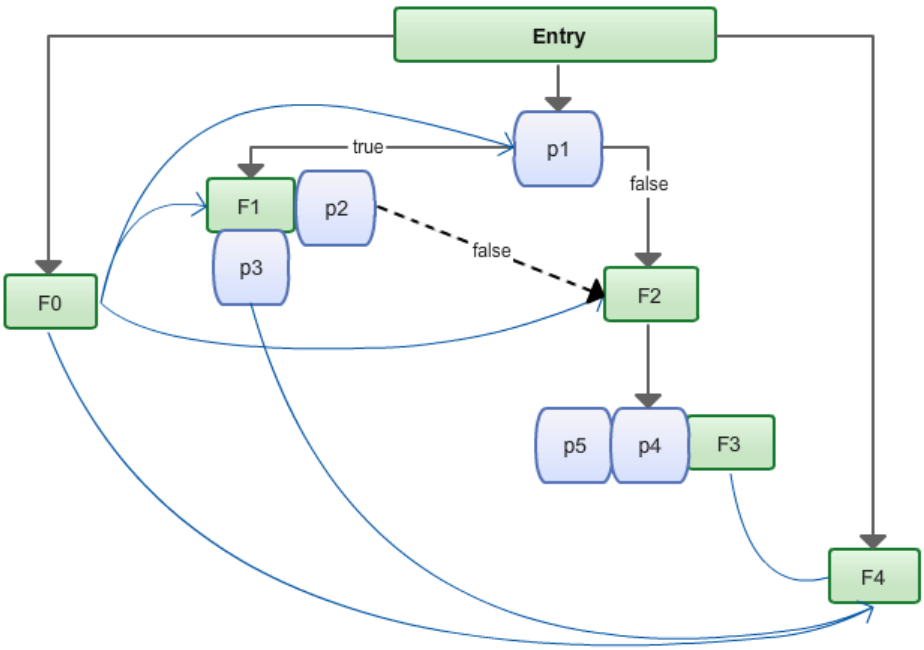

Abbildung 16: Program Dependence Graph, Schritt 2

Als letzte mögliche Zusammenlegung machen wir aus p1 und F0 ein nicht portables Fragment. Beide haben den gleichen Elternknoten und p1 ist datenabhängig von F0.
Danach ist der PDG schon ziemlich kompakt, da alle Datenflussabhängigkeiten (blaue Pfeile) weggelassen werden können, die nur dem Kontrollfluss folgen (F0->p1, F0->{F1, p2, p3} und F0->F2).

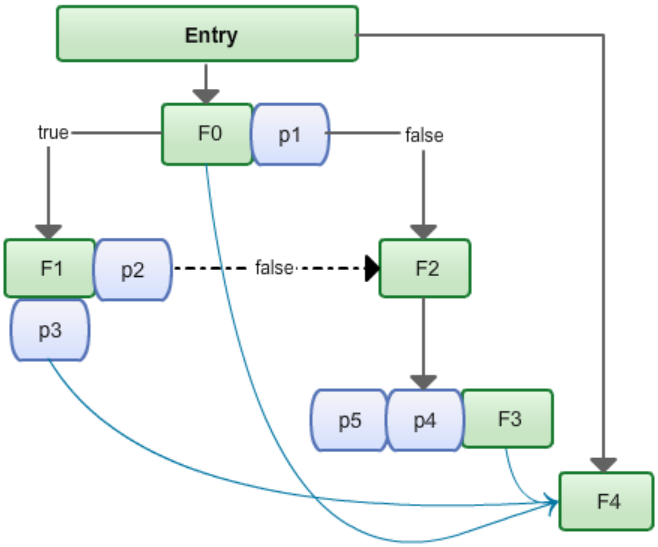

Abbildung 17: Program Dependence Graph, Schritt 3

Nun sind keine weiteren Zusammenlegungen möglich, da F4 ein festes Fragment ist, das keine portablen Kinder oder Eltern hat, mit denen man ein Fragment bilden könnte.
F2 und {p5, p4, F3} können nicht zusammengelegt werden, da zwei nicht portable Aktivitäten (F2 und F3) nicht im selben Fragment sein können.

6 Fazit

Ziel dieser Arbeit war es, verschiedene Ansätze zur Fragmentierung von Geschäftsprozessen miteinander zu vergleichen. Unserer Meinung nach liefert die nun vorliegende Fachstudie einen guten Überblick über die aktuell in der Forschung vorhandenen Ansätze. Sie bietet ausführliche Zusammenfassungen von Zielen, Fähigkeiten und Problemen jedes Ansatzes und ermöglicht weiteren Arbeiten in diesem Themengebiet, auf ihr aufzubauen.

Als sehr interessant erwiesen sich für uns die unterschiedlichen Herangehensweisen an die Fragmentierung, in denen sich die große Bandbreite von Anwendungsgebieten und Zielen spiegelten. Auch das Fragment selbst tritt in unterschiedlichen Formen auf -- sei es als Menge von Aktivitäten, als SESE-Fragment, Stratum oder Baustein für einen neuen Geschäftsprozess.

Eine grundsätzliche Schwierigkeit bestand darin, dass die Ansätze in Ziel und Art derart unterschiedlich sind, dass ein direkter Vergleich mit Bestimmung des besten Ansatzes uns nicht möglich erschien. Ursprünglich war hierzu ein Kriterienkatalog mit Punktevergabe angedacht, was sich aber letztendlich als nicht realisierbar herausstellte. Ein wirklicher Vergleich ist unserer Meinung nach nur möglich, wenn Fragmentierungs-Ansätze für ein bestimmtes, eingeschränktes Ziel (beispielsweise verteilte Ausführung zur Erhöhung des Durchsatzes) gesucht werden.

7 Abbildungsverzeichnis

Abbildung 1: Tagcloud der am häufigsten verwendeten Wörter (wordle.net) 2
Abbildung 2: Operations-Szenario des Fragment-gesteuerten Ansatzes (Quelle: [6]) 14
Abbildung 3: Prozessmodell "LoanApprovalExtended" 32
Abbildung 4: Fragmentierung des Prozessmodells 33
Abbildung 5: Wiederhergestellter Link (Quelle: [1], Figure 8) 35
Abbildung 6: Grafische Darstellung der Fragmentierung 36
Abbildung 7: Hauptfragment 37
Abbildung 8: Risk Assessor/Address Checker 37
Abbildung 9: Loan Approval 37
Abbildung 10: Dependence Graph 38
Abbildung 11: Stratification Graph 39
Abbildung 12: Threaded Control Flow Graph 41
Abbildung 13: Program Dependence Graph 42
Abbildung 14: Program Dependence Graph, Ausgangszustand 42
Abbildung 15: Program Dependence Graph, Schritt 1 43
Abbildung 16: Program Dependence Graph, Schritt 2 43
Abbildung 17: Program Dependence Graph, Schritt 3 44

8 Tabellenverzeichnis

Tabelle 1: In welchem Format muss das Prozessmodell beim jeweiligen Ansatz vorliegen 31

9 Literaturverzeichnis

[1] *Role-based Decomposition of Business Processes using BPEL.* **Khalaf, Rania und Leymann, Frank.** 2006. International Conference on Web Services.

[2] *A Novel Approach to Decentralized Workflow Enactment.* **Martin, Daniel, Wutke, Daniel und Leymann, Frank.** München : s.n., 2008. Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference.

[3] *Optimal Stratification of Transactions.* **Danylevych, Olha, Karastoyanova, Dimka und Leymann, Frank.** 2009. ICIW.

[4] *BPEL Fragments for Modularized Reuse in Modeling BPEL Processes.* **Ma, Zhilei und Leymann, Frank.** Valencia : s.n., 2009. The Fifth International Conference on Networking and Services.

[5] *On Application of Structural Decomposition for Process Model Abstraction.* **Polyvyanyy, Artem, Smirnov, Sergey und Weske, Mathias.** Leipzig : s.n., 2009. Proceedings of the 2nd International Conference on Business Process and Services Computing.

[6] *A Fragment-Driven Process Modeling Methology.* **Kim, Kwang-Hoon, Won, Jae-Kang und Kim, Chang-Min.** Singapore : s.n., 2005. Computational Science and Its Applications.

[7] *Modelling inter-organizational processes with process model fragments.* **Lindert, Frank und Deiters, Wolfgang.** 1999. Proceedings of Workshop Informatik ’99: Enterprise-wide and Crossenterprise Workflow Management: Concepts, Systems, Applications. S. 33–41.

[8] *Supporting Workflow Cooperation within and across Organizations.* **Casati, Fabio und Discenza, Angela.** Como, Italy : s.n., 2000. SAC 2000.

[9] *Decentralizing execution of composite Web services.* **Nanda, Mangala Gowri, Chandra, Satish und Sarkar, Vivek.** 2004. OOPSLA. S. 170-187.

[10] *Distributed Workflow Execution based on fragmentation of Petri Nets.* **Guth, Volker, Lenz, Kirsten und Oberweis, Andreas.** Vienna : OCG, Vienna, AUTRICHE, 1998. IFIP world computer congress No15. S. 114-125.

[11] **Tan, Wei und Fan, Yushun.** Dynamic workflow model fragmentation for distributed execution. *Computers in Industry.* 2007, Bd. 58, 5, S. 381-391.

[12] *Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition.* **Vanhatalo, Jussi, Völzer, Hagen und Leymann, Frank.** Vienna : s.n., 2007. Proceedings of the 5th international conference on Service-Oriented Computing. S. 43-55.

[13] *The Refined Process Structure Tree.* **Vanhatalo, Jussi, Völzer, Hagen und Koehler, Jana.** Milan, Italy : s.n., 2008. Proceedings of the 6th International Conference on Business Process Management. S. 100-115.

[14] *Automatic Workflow Graph Refactoring and Completion.* **Vanhatalo, Jussi, et al.** Sydney, Australia : s.n., 2008. Proceedings of the 6th International Conference on Service-Oriented Computing. S. 100-115.

[15] **Khalaf, Rania.** *Supporting Business Process Fragmentation While Maintaining Operational Semantics: A BPEL Perspective.* Stuttgart : s.n., 2008.

[16] **Oasis.** Web Services Business Process Execution Language Version 2.0. [Online] 11. 04 2007. [Zitat vom: 21. 09 2009.] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[17] **Khalaf, Rania.** *Note on Syntactic Details of Split BPEL-D Business Processes.* Stuttgart : s.n., 2007.

[18] *Model and infrastructure for decentralized workflow enactment.* **Wutke, Daniel, Martin, Daniel und Leymann, Frank.** New York : ACM, 2008. Proceedings of the 2008 ACM symposium on Applied computing. S. 90-94.

[19] *A method for partitioning BPEL processes for decentralized execution.* **Wutke, Daniel, Martin, Daniel und Leymann, Frank.** Stuttgart : s.n., 2009. 1st Central-European Workshop on Services and their Composition, Proceedings 2009.

[20] **Wu, Shenqiang.** *Verteilte Workflow-Engine: Ausführung verteilter Prozessmodelle auf Basis von Tuplespaces (Diplomarbeit).* Stuttgart : s.n., 2008.

[21] **Nanda, Mangala Gowri, Chandra, Satish und Sarkar, Vivek.** Decentralizing CompositeWeb Services.

10 Anhang

10.1 LoanApprovalExtended.bpel

```
<process name="LoanApprovalExtended"
   targetNamespace="http://example.com/loan-approval/"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:lns="http://example.com/loan-approval/wsdl/"
   suppressJoinFailure="yes"                 xmlns:bpws="http://docs.oasis-
open.org/wsbpel/2.0/process/executable">

 <!-- #################### IMPORT #################### -->
   <import importType="http://schemas.xmlsoap.org/wsdl/"
      location="LoanApprovalExtended.wsdl"
      namespace="http://example.com/loan-approval/wsdl/" />

 <!-- #################### PARTNER LINKS #################### -->
   <partnerLinks>
      <partnerLink name="customer"
        partnerLinkType="lns:loanPartnerLT"
        myRole="loanService" />
      <partnerLink name="approver"
        partnerLinkType="lns:loanApprovalLT"
        partnerRole="approver" />
      <partnerLink name="assessor"
        partnerLinkType="lns:riskAssessmentLT"
        partnerRole="assessor" />
         <partnerLink name="reviewer"
        partnerLinkType="lns:addressCheckLT"
        partnerRole="reviewer" />
   </partnerLinks>

 <!-- #################### VARIABLES #################### -->
   <variables>
      <variable  name="request"  messageType="lns:creditInformationMessage"
/>
      <variable name="risk" messageType="lns:riskAssessmentMessage" />
      <variable name="approval" messageType="lns:approvalMessage" />
      <variable                                          name="addressCheck"
messageType="lns:addressCheckResultMessage" />
   </variables>

 <!-- #################### FAULT HANDLER #################### -->
   <faultHandlers>
      <catch faultName="lns:loanProcessFault"
        faultVariable="error"
        faultMessageType="lns:errorMessage">
         <reply partnerLink="customer"
           portType="lns:loanServicePT"
           operation="request" variable="error"
           faultName="unableToHandleRequest" />
      </catch>
   </faultHandlers>

   <flow>
   <!-- #################### LINKS #################### -->
      <links>
         <link name="receive-to-assess" />
         <link name="assess-to-assignYes" />
```

```
            <link name="assignYes-to-reply" />

            <link name="assess-to-addresscheck" />
            <link name="receive-to-addresscheck" />

            <link name="addresscheckYes-to-approval" />
            <link name="approval-to-reply" />

            <link name="addresscheckNo-to-assignNo" />
            <link name="assignNo-to-reply" />
        </links>

    <!-- #################### ReceiveRequest ############################## -
->
        <receive partnerLink="customer"
          portType="lns:loanServicePT"
          operation="request"
          variable="request"
          createInstance="yes"
          name="ReceiveRequest">
           <sources>
              <source linkName="receive-to-assess">
                 <transitionCondition>
                    $request.amount &lt; 10000
                 </transitionCondition>
              </source>
              <source linkName="receive-to-addresscheck">
                 <transitionCondition>
                    $request.amount >= 10000
                 </transitionCondition>
              </source>
           </sources>
        </receive>

    <!-- #################### InvokeRiskAssessment #################### -->
        <invoke partnerLink="assessor"
          portType="lns:riskAssessmentPT"
          operation="check"
          inputVariable="request"
          outputVariable="risk"
          name="InvokeRiskAssessment">
           <targets>
              <target linkName="receive-to-assess" />
           </targets>
           <sources>
              <source linkName="assess-to-assignYes">
                 <transitionCondition>
                    $risk.level='low'
                 </transitionCondition>
              </source>
              <source linkName="assess-to-addresscheck">
                 <transitionCondition>
                    $risk.level!='low'
                 </transitionCondition>
              </source>
           </sources>
        </invoke>
```

```
<!-- #################### AssignYes #################### -->
   <assign name="AssignYes">
      <targets>
         <target linkName="assess-to-assignYes" />
      </targets>
      <sources>
         <source linkName="assignYes-to-reply" />
      </sources>
      <copy>
         <from>
            <literal>yes</literal>
         </from>
         <to variable="approval" part="accept" />
      </copy>
   </assign>

<!-- #################### InvokeAddressCheck #################### -->
    <invoke partnerLink="reviewer"
     portType="lns:addressCheckPT"
     operation="checkAddress"
     inputVariable="request"
     outputVariable="addressCheck"
     name="InvokeAddressCheck">
      <targets>
         <target linkName="receive-to-addresscheck" />
         <target linkName="assess-to-addresscheck" />
      </targets>
      <sources>
         <source linkName="addresscheckYes-to-approval">
            <transitionCondition>
               $addressCheck.correct='yes'
            </transitionCondition>
         </source>
         <source linkName="addresscheckNo-to-assignNo">
            <transitionCondition>
               $addressCheck.correct='no'
            </transitionCondition>
         </source>
      </sources>
   </invoke>

 <!-- #################### AssignNo #################### -->
   <assign name="AssignNo">
      <targets>
         <target linkName="addresscheckNo-to-assignNo" />
      </targets>
      <sources>
         <source linkName="assignNo-to-reply" />
      </sources>
      <copy>
         <from>
            <literal>no</literal>
         </from>
         <to variable="approval" part="accept" />
      </copy>
   </assign>

<!-- #################### InvokeLoanApproval #################### -->
   <invoke partnerLink="approver"
     portType="lns:loanApprovalPT"
     operation="approve"
```

```
        inputVariable="request"
        outputVariable="approval"
        name="InvokeLoanApproval">
         <targets>
            <target linkName="addresscheckYes-to-approval" />
         </targets>
         <sources>
            <source linkName="approval-to-reply" />
         </sources>
      </invoke>

   <!-- #################### ReplyResult #################### -->
      <reply partnerLink="customer"
        portType="lns:loanServicePT"
        operation="request"
        variable="approval"
        name="ReplyResult">
         <targets>
            <target linkName="assignYes-to-reply" />
            <target linkName="approval-to-reply" />
            <target linkName="assignNo-to-reply" />
         </targets>
      </reply>
       </flow>
</process>
```

10.2 LoanApprovalExtended.wsdl

```
<wsdl:definitions
   targetNamespace="http://example.com/loan-approval/wsdl/"
   xmlns:ens="http://example.com/loan-approval/xsd/error-messages/"
   xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
   xmlns:tns="http://example.com/loan-approval/wsdl/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema/">

<!-- #################### MESSAGES #################### -->
   <wsdl:message name="creditInformationMessage">
      <wsdl:part name="firstName" type="xsd:string" />
      <wsdl:part name="lastName" type="xsd:string" />
      <wsdl:part name="address" type="xsd:string" />
      <wsdl:part name="amount" type="xsd:integer" />
   </wsdl:message>

   <wsdl:message name="approvalMessage">
      <wsdl:part name="accept" type="xsd:string" />
   </wsdl:message>

   <wsdl:message name="addressCheckResultMessage">
      <wsdl:part name="correct" type="xsd:string" />
   </wsdl:message>

   <wsdl:message name="riskAssessmentMessage">
      <wsdl:part name="level" type="xsd:string" />
   </wsdl:message>

   <wsdl:message name="errorMessage">
      <wsdl:part name="errorCode" element="ens:integer" />
   </wsdl:message>

<!-- #################### PORT TYPES #################### -->
   <wsdl:portType name="loanServicePT">
```

```
      <wsdl:operation name="request">
         <wsdl:input message="tns:creditInformationMessage" />
         <wsdl:output message="tns:approvalMessage" />
         <wsdl:fault                          name="unableToHandleRequest"
message="tns:errorMessage" />
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:portType name="riskAssessmentPT">
      <wsdl:operation name="check">
         <wsdl:input message="tns:creditInformationMessage" />
         <wsdl:output message="tns:riskAssessmentMessage" />
         <wsdl:fault name="loanProcessFault" message="tns:errorMessage" />
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:portType name="loanApprovalPT">
      <wsdl:operation name="approve">
         <wsdl:input message="tns:creditInformationMessage" />
         <wsdl:output message="tns:approvalMessage" />
         <wsdl:fault name="loanProcessFault" message="tns:errorMessage" />
      </wsdl:operation>
   </wsdl:portType>

       <wsdl:portType name="addressCheckPT">
      <wsdl:operation name="checkAddress">
         <wsdl:input message="tns:creditInformationMessage" />
         <wsdl:output message="tns:addressCheckResultMessage" />
         <wsdl:fault name="loanProcessFault" message="tns:errorMessage" />
      </wsdl:operation>
   </wsdl:portType>

<!-- #################### PARTNER LINK TYPES #################### -->
   <plnk:partnerLinkType name="loanPartnerLT">
      <plnk:role name="loanService" portType="tns:loanServicePT" />
   </plnk:partnerLinkType>

   <plnk:partnerLinkType name="loanApprovalLT">
      <plnk:role name="approver" portType="tns:loanApprovalPT" />
   </plnk:partnerLinkType>

   <plnk:partnerLinkType name="riskAssessmentLT">
      <plnk:role name="assessor" portType="tns:riskAssessmentPT" />
   </plnk:partnerLinkType>

   <plnk:partnerLinkType name="addressCheckLT">
      <plnk:role name="reviewer" portType="tns:addressCheckPT" />
   </plnk:partnerLinkType>

</wsdl:definitions>
```

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

Ort, Datum Tobias Binz

_____

Ort, Datum Jan Königsberger

_____

Ort, Datum Tina Schliemann