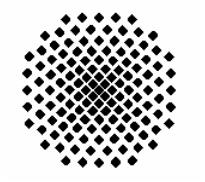

Universität Stuttgart
Fakultät Informatik, Elektrotechnik und Informationstechnologie

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Uwe Heinkel
Beginn am:	21. Oktober 2003
Beendet am:	21. April 2004
CR-Classifikation:	C.4, C.2.4

Diplomarbeit Nr. 2155

Leistungsmessung und Leistungssteigerung des Propagationssystems

Gueorgui Ovtcharov

Institut für Parallele und Verteilte Systeme (IPVS)
Abteilung Anwendersoftware

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Kurzfassung

Das Aufgabengebiet des Teilprojekts A5 des SFB 467 „Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion“ ist die informationstechnische Unterstützung wandlungsfähiger Unternehmen. In diesem Zusammenhang wurde ein Prototyp, das „Stuttgarter Informations- und Explorationssystem“ (SIES), entwickelt, dessen Aufgabe ist es, Datenabhängigkeiten zwischen heterogenen Systemen zu verwalten und Datenänderungen zu propagieren. Die Autonomie der einzelnen Systeme wird dabei gewährleistet.

Herzstück des SIES ist der Propagationsmanager. Er ermöglicht die Weiterleitung von Datenänderungen eines Informationssystems an die weiteren abhängigen Informationssysteme.
Die Leistungssteigerung des Propagationsmanagers ist das Ziel dieser Arbeit. Um die Leistung des Systems verbessern zu können, muss man in der Lage sein, die Leistung des Systems zu messen. In dieser Arbeit werden eine Testumgebung und Testfälle entwickelt, mit deren Hilfe die Leistungsmessung des Propagationsmanagers ermöglicht wird. Anhand der gewonnen Messdaten werden Optimierungsmaßnahmen zur Leistungsverbesserung des Propagationsmanagers diskutiert und umgesetzt. Zum Schluss werden weitere Konzepte zur Leistungssteigerung des Propagationsmanagers diskutiert, die im Rahmen dieser Arbeit nicht umgesetzt wurden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Hintergrund

Die Dynamik und Komplexität der Unternehmensumwelt haben in den letzten Jahrzehnten ständig und drastisch zugenommen. Eine Definition dieser zwei Begriffe wird in [HP00] gegeben: „Die Umweltdynamik ist charakterisiert durch die Häufigkeit und Geschwindigkeit von Veränderungen einzelner Umweltsegmente, durch die Stärke dieser Veränderungen und durch die Regelmäßigkeit und Vorhersehbarkeit der Veränderungen. Die Komplexität der Umwelt kann beschrieben werden durch die Anzahl und Verschiedenartigkeit der für die Unternehmung relevanten Umwelttatbestände“.
Die Dynamik und Komplexität der Unternehmensumwelt wirkt sich direkt auf die Organisationsstruktur der Unternehmen aus. Eine zunehmende organisatorische Differenzierung (überstark arbeitsteilige Organisationsstruktur) innerhalb der Unternehmen ist die Folge dieser Faktoren. Diese Differenzierung führt zu Koordinationsproblemen innerhalb des Unternehmens, die durch umfassende Integrationsmaßnahmen gelöst werden.

Die Herausforderungen des Wettbewerbs haben in den vergangenen Jahren zu einem grundlegenden und umfassenden Wandel in der Gestaltung der Unternehmensstrukturen geführt. Zwei Ansätze haben in den 90-er Jahren des letzten Jahrhunderts besonders an Bedeutung gewonnen:

- „Lean management“ : Vereinfachung von Strukturen zwecks effektiver und effizienterer Führung.
- „Business Process Reenginieering“ : Radikales Aufheben der überstarken Arbeitsteilung und hierdurch Schaffung einer Kundenorientierung.

Die Entwicklungen im Gebiet der Informationstechnologien spielen bei dem Prozess des Wandels eine große Rolle. Der Einsatz neuer Informationstechnologien verbessert den Informationsfluss nicht nur innerhalb eines Unternehmens sondern auch zwischen mehreren Unternehmen. Auf diese Weise können Wettbewerbsvorteile geschaffen werden.

1.2 Sonderforschungsbereich 467

Die Zielsetzung des Sonderforschungsbereichs (SFB) 467 „wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion“ ist die „Erarbeitung eines theoretisch fundierten, fachübergreifend überprüften und umsetzbaren Grundkonzepts für wandlungsfähige Unternehmen“ [SFB03].

Die Fähigkeit eines Unternehmens Veränderungen in der Unternehmensumwelt frühzeitig zu erkennen und sich an diese Veränderungen anzupassen, wird als *Wandlungsfähigkeit* bezeichnet. Veränderungen der Unternehmensumwelt können für das Unternehmen nicht nur Risiken und Bedrohungen, sondern auch neue Chancen bedeuten.

Der im SFB verfolgte Ansatz der Wandlungsfähigkeit betrachtet das System des Unternehmens als ein Netzwerk kommunizierender, *autonomer Leistungseinheiten*. Diese sind in der Lage, interne Strukturen, Ressourcen und Prozesse eigenständig und ohne äußere Eingriffe optimiert auszurichten.
Leitungseinheiten können weitere untergeordnete Leistungseinheiten beinhalten. Die Führung einer Leistungseinheit bestimmt die Rahmenbedingungen, legt die Ziele und Verhaltensregeln im Netz der untergeordneten Einheiten fest und kontrolliert deren Einhaltung [KO00]. Die Selbstorganisation der einzelnen Einheiten wird als Schlüsselfaktor für die Anpassungsfähigkeit des gesamten Unternehmens gesehen. Sie sind in der Lage die Veränderungen der Umwelt zu analysieren und sich schnell an diese Veränderungen anzupassen.
Die Anpassung an die Unternehmensumwelt soll dabei auf die Ziele des gesamten Unternehmens ausgerichtet sein. Die Gesamtziele des Unternehmens sind die Rahmenbedingungen, die den Verhaltensraum der einzelnen Leistungseinheiten definieren. Aus diesem Grund spricht man auch von *Teilautonomie* der Leistungseinheiten [HU00].

Der Sonderforschungsbereich 467 wurde 1997 in Stuttgart initiiert und verläuft in drei Förderperioden. In der ersten Förderperiode, die den Zeitraum von 1997 bis 1999 umfasst, wurden theoretische Grundlagen und Modelle zur Erreichung der oben definierten Ziele entwickelt und in der zweiten Phase (von 2000 bis 2002) wurde ein Gesamtkonzept erarbeitet. Im Rahmen der gegenwärtigen, dritten Förderperiode von 2003 bis 2005, sollen die entstandenen Teillösungen vollständig integrieren werden.

Die Arbeit im SFB 467 verläuft in mehreren Teilprojekten, die miteinander kooperieren. Das für die vorliegende Diplomarbeit relevante Teilprojekt A5 wird im Folgenden vorgestellt.

1.3 Teilprojekt A5

Das Ziel des Teilprojekts A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ ist die informationstechnische Unterstützung wandlungsfähiger Unternehmen in der variantenreichen Serienproduktion.

Ein Schwerpunkt des Teilprojekts A5 ist die Modellierung der komplexen Informationsstruktur in wandlungsfähigen Unternehmen. Dabei werden Datenmodelle

für die einzelnen Informationssysteme entwickelt. So wurden im Rahmen der Teilprojekte B2 „Dynamische Fabrikstrukturen" und C1 „Kooperative Planung und Steuerung in dynamischen Prozessketten" für zwei wichtige Bereiche im Unternehmen, das Auftragmanagement und die Fabriklayoutplanung, Datenmodelle definiert.

Diese Modelle sind zunächst voneinander isoliert und müssen zu einem ganzheitlichen Unternehmensmodell integriert werden. Bei der Integration der einzelnen Informationssysteme ist man von einem föderativen Ansatz herausgegangen, bei dem die Autonomie der einzelnen Systeme gewährleistet wird. Wenn ein System an Datenänderungen eines anderen Systems interessiert ist, dann besteht eine Abhängigkeit zwischen diesen Systemen. Falls eine Datenänderung erfolgt, müssen alle abhängigen Systeme darüber benachrichtigt werden. Das „Stuttgarter Informations- und Explorationssystem" (SIES) ist eine prototypische Umsetzung dieses Konzepts.

Das SIES besteht aus zwei unabhängigen Systemen, dem Abhängigkeitsmanager und dem Propagationsmanager. Der Abhängigkeitsmanager ermöglicht die Verwaltung der Datenmodelle der Informationssysteme und die Verwaltung der Datenabhängigkeiten zwischen den Informationssystemen. Die Weiterleitung der Datenänderungen wird durch den Propagationsmanager realisiert. Auf die Architektur des SIES wird in Kapitel 2 näher eingegangen.

1.4 Zielsetzung der Diplomarbeit

Zielsetzung dieser Arbeit ist die Leistungserhebung und Leistungssteigerung des Propagationsmanagers. Um dieses Ziel zu erreichen wird die Arbeit in folgende Teilaufgaben gegliedert:

- Entwicklung einer Testumgebung, mit deren Hilfe die Leistung des Propagationsmanagers und seiner Komponenten gemessen werden kann.

- Erstellung unterschiedlicher Tests, die zur Untersuchung der Leistung des Propagationsmanagers verwendet werden. Die Ergebnisse der durchgeführten Tests sollen die Schwachstellen des Propagationssystems aufzeigen und eine Entscheidungsgrundlage zur Leistungsverbesserung des untersuchten Systems liefern.

- Leistungserfassung des Propagationsmanagers und seiner Komponenten anhand der definierten Tests und Erarbeitung von Verbesserungsvorschlägen zur Leistungssteigerung des Systems.

1.5 Gliederung der Arbeit

Um einen Überblick über die vorliegende Arbeit zu geben, werden im Folgenden die Inhalte der einzelnen Kapitel kurz beschrieben.

In *Kapitel 2* werden die Architektur des „Stuttgarter Informations- und Explorationssystem" und die Komponenten des Propagations- und

Abhängigkeitsmanagers vorgestellt. Weiterhin werden die entscheidenden Faktoren für seine Leistungsfähigkeit untersucht.

Das im Rahmen dieser Diplomarbeit verwendete Konzept zur Leistungsmessung und Leistungsanalyse wird in *Kapitel 3* beschrieben. Weiterhin werden die Architektur der Testumgebung, die Messmethoden und die Messgrößen vorgestellt. Als Nächstes werden die funktionalen und nicht-funktionalen Anforderungen an die Testumgebung definiert.

Kapitel 4 beschäftigt sich mit dem Entwurf und der Implementierung der Testumgebung. Zunächst werden die Komponenten der zu entwickelnden Anwendung, ihre Funktionsweise und ihr Zusammenwirken beschrieben. Danach wird auf den Feinentwurf der einzelnen Komponenten eingegangen. Schließlich wird die Implementierung der Testumgebung beschrieben und die Installation, die Konfiguration und die Bedienung des entwickelten Tools werden erläutert.

In *Kapitel 5* werden die Testfälle, die zur Untersuchung der Leistung des Propagationsmanagers verwendet werden, definiert. Daraufhin werden die Messergebnisse präsentiert, die die Leistung des Propagationsmanagers zum Beginn dieser Diplomarbeit widerspiegeln.

Ausgehend von den Ergebnissen der am Propagationssystem durchgeführten Messungen werden in *Kapitel 6* Optimierungsmaßnahmen für den Propagationsmanager getroffen und schließlich umgesetzt. Des Weiteren werden die umgesetzten Optimierungsmaßnahmen anhand von Messungen bewertet.

In *Kapitel 7* werden weitere wichtige Konzepte zur Leistungssteigerung des Propagationsmanagers diskutiert, die aus zeitlichen Gründen im Rahmen dieser Arbeit nicht umgesetzt werden konnten.

Schließlich werden die Ergebnisse dieser Arbeit in *Kapitel 8* zusammengefasst.

Kapitel 2

Stuttgarter Informations- und Explorationssystem (SIES)

2.1 Einführung

Als Lösung der im vorigen Kapitel vorgestellten Problematik ist im Rahmen des Teilprojekts A5 „Modellierung von und Exploration in komplexen Unternehmensinformationen“ das SIES (*Stuttgarter Informations- und Explorationssystems*) entstanden. Bei diesem Ansatz bleibt die Autonomie der einzelnen Informationssysteme innerhalb des Unternehmens bestehen (das Konzept schließt die Einbeziehung weiterer Informationssysteme von außerhalb des Unternehmens nicht aus). Ein System kann hier ein Informationssystem oder eine Anwendung sein, wobei ein Unterschied zwischen *Quellsystemen* (Datenproduzenten), und *Zielsystemen* (Datenkonsumenten) gemacht wird. Ein System kann hierbei gleichzeitig Quell- und Zielsystem sein.

Die grundlegende Idee des SIES-Ansatzes ist es, Datenänderungen von einem oder mehreren Quellsystemen an ein oder mehrere Zielsysteme weiterzuleiten. Unter Datenänderung wird eine Einfüge-, Lösch- oder Update-Operation verstanden. Es bestehen Beziehungen zwischen den Quell- und den Zielsystemen, die als *Datenabhängigkeiten* oder einfach als *Abhängigkeiten* bezeichnet werden. Je nach Anzahl der Quellsysteme und Zielsysteme unterscheidet man drei Typen von Datenabhängigkeiten:

- *1:1*: Ein Quellsystem und ein Zielsystem,
- *1:n*: Ein Quellsystem und *n* Zielsysteme,
- *m:n*: *m* Quellsysteme und *n* Zielsysteme.

Der Vorgang des Weiterleitens einer Datenänderung an alle abhängigen Systeme wird als *Datenänderungspropagation*, bezeichnet.

Das Einsatzgebiet des SIES schließt die Integration heterogener Systeme mit unterschiedlichen Datenmodellen ein. Aus diesem Grund müssen die Datenänderungen eines Quellsystems in das entsprechende Format des Zielsystems gebracht werden. Die Operation, die diese Umwandlung durchführt, wird als *Transformation* bezeichnet.

Innerhalb des SIES wurde eine weitere Operation, die *Filterung*, eingeführt. Hierbei wird getestet, ob die Daten einen booleschen Ausdruck erfüllen. Das Ergebnis dieser Auswertung bestimmt den weiteren Ablauf einer Propagation, wodurch bedingte Verzweigungen realisiert werden können [HM02].

Im nächsten Abschnitt wird die Integration heterogener Informationssystem auf Basis der XML-Technologie vorgestellt und die Zusammenhänge zwischen den oben genannten Begriffen verdeutlicht.

2.2 Einsatz von XML-Technologien im SIES

2.2.1 Datenpräsentation

Die *Extensible Markup Language (XML)* [W3C00] hat sich in den letzten Jahren als Standardformat für den Datenaustausch etabliert. Für die Anforderungen des Propagationsmanagers, die Integration heterogener Informationssysteme, ist XML gut geeignet. Die Datenänderungen eines Informationssystems werden in ein XML-Dokument gespeichert, das gemäß einer Spezifikation in ein oder mehrere XML-Dokumente umgewandelt wird. Dieser Umwandlungsprozess ist in Abbildung 2.1 dargestellt. Die transformierten XML-Dokumente werden schließlich an die abhängigen Zielsysteme weitergeleitet.

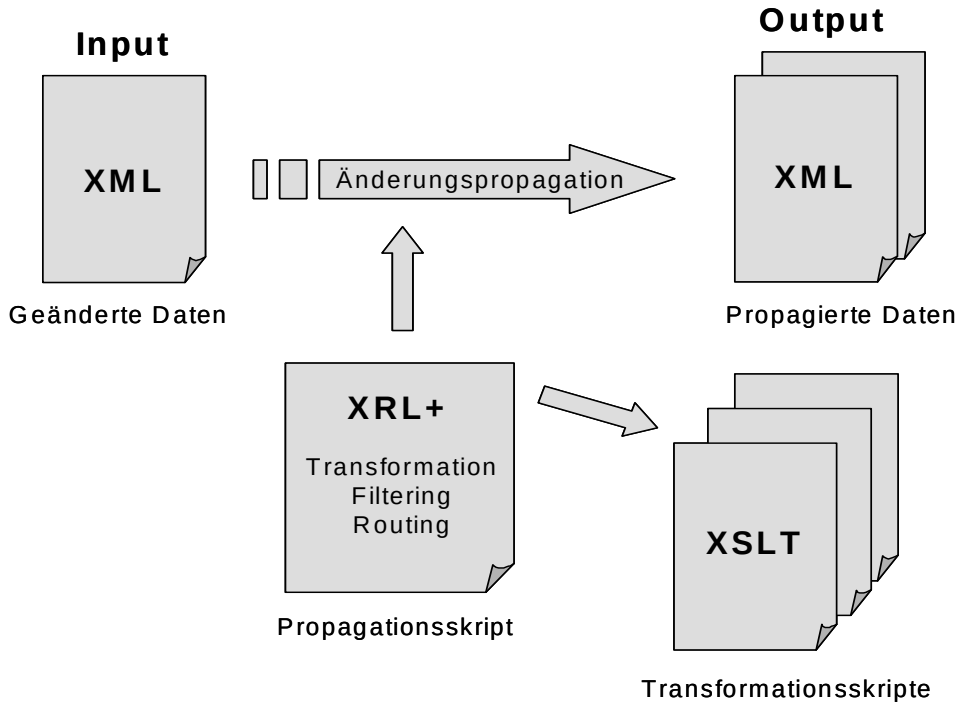

Abbildung 2.1: Einsatz von XML-Technologien im SIES

Der nächste Abschnitt beschäftigt sich mit der Spezifikation der Datenänderungspropagationen im Rahmen des SIES.

2.2.2 Transformation und Filterung

In einem heterogenen Umfeld unterscheiden sich die Datenrepräsentationen der unterschiedlichen Informationssysteme häufig in ihrer Struktur. Deswegen sind Transformationen der propagierenden Datenänderungen vom Format des Quellsystems in die Formate der Zielsysteme notwendig. Eine Transformation kann folgende Umwandlungsoperationen einschließen [KO01]:

- *Typumwandlungen* sind erforderlich, wenn die Datenrepräsentation von System zu System variiert. Ein Beispiel dafür ist die Umwandlung boolescher Werte von textueller (*„true“* und *„false“*) in numerische Darstellung (*1* und *0*).

- Das *Erzeugen* und *Aufbrechen* von Datenstrukturen ist eine der am häufigsten angewendeten Transformationen im Propagationssystem.

- *Funktionale Transformationen* und *Berechnungen* können angewendet werden, um zum Beispiel die Daten zu aggregieren.

- Durch *Integration verschiedener Datenquellen* ist es möglich, Daten aus verschiedenen Quellsystemen zu kombinieren und das Ergebnis an ein oder mehrere Zielsysteme zu senden.

2.2.2.1 XSLT

Ursprünglich wurde XSLT zur Transformation von XML-Daten in HTML-Datenformat verwendet. Mittlerweile hat sich XSLT für Transformation von XML-Dokumenten etabliert.
Ein XSLT-Skript beschreibt Regeln für die Transformation des Quellbaums eines XML-Dokuments in einen Ergebnisbaum. Diese Transformation wird durch die Assoziation von *Mustern* mit *Templates* erreicht. Ein Muster wird gegen die Elemente des Quellbaums getestet. Ein Template beschreibt einen Teil des Ergebnisbaums. Die Struktur des Ergebnisbaums kann sich von der Struktur des Quellbaums komplett unterscheiden. Bei der Konstruktion des Ergebnisbaums können Elemente des Quellbaums gefiltert und umgeordnet werden. Zum Beispiel können Elemente weggelassen oder neu hinzugefügt werden oder eine Sortierung der Elemente kann vorgenommen werden [MS02].

2.2.2.2 XPath

XPath [W3C99] ist das Ergebnis der Bemühungen, eine gemeinsame Syntax und Semantik für jene Funktionen bereitzustellen, die sowohl von XSL Transformationen als auch von XPointer genutzt werden. Die primäre Aufgabe von XPath besteht in der Adressierung von Teilen eines XML-Dokuments. Zur Unterstützung dieser Aufgabe werden außerdem einfache Hilfsmittel für die Manipulation von Zeichenketten, Zahlen und booleschen Werten bereitgestellt.

2.2.3 Definition der Datenabhängigkeiten

Die Definition der Datenabhängigkeiten zwischen den Quell- und Zielsystemen ist ein zentrales Thema im SIES. Für diese Zwecke werden Metadaten in einem Repository (siehe Abschnitt 2.3.3) verwaltet. In diesem Abschnitt wird die *XRL+* (eXchangeable Routing Language Plus) vorgestellt, die den Ablauf der Datenänderungspropagationen im Propagationsmanager definiert. Ausführliche Dokumentation über die Modellierung der Datenabhängigkeiten im SIES kann der neugierige Leser in [MH02] finden.

Die Propagationsbeschreibungssprache XRL+ basiert auf basiert auf der Workflowdefinitionssprache eXchangeable Routing Language (XRL) [AW00] und wurde für die Anforderungen des Propagationsmanagers angepasst.

Das Element <*xrl:ROUTE*> ist das Wurzelelement jedes XRL+-Dokuments. Durch die Attribute *id*, *created_by* und *creation_date* können zusätzliche Informationen für die Dokumente definiert werden.

Der Kontrollfluss der Änderungspropagationen wird durch folgende sechs Elemente der XRL+ beschrieben:

- <*xrl:SEQUENCE*> definiert eine sequenzielle Reihenfolge der Ausführung seiner untergeordneten Elemente.

- Durch das Element <*xrl:PARALLEL*> können untergeordnete Elemente parallel zueinander ausgeführt werden. Das Attribut *sync* spezifiziert die Anzahl der Elemente, die beendet werden, bevor die Ausführung der Änderungspropagation fortgesetzt wird. Dadurch kann eine einfache Synchronisation der parallel ausgeführten Elemente erreicht werden.

- <*xrl:CONDITION*> entspricht einer *if-else*-Anweisung. Die beiden Zweige werden durch die Unterelemente <*xrl:true*> und <*xrl:false*> gekennzeichnet. Die Bedingung dieses Elements ist ein *XPath-Ausdruck*, der durch das Attribut *expression* definiert wird. Das XML-Dokument wird durch das Attribut *xml* referenziert. Falls der Ausdruck als *true* ausgewertet ist, werden die Schritte im Unterelement <*xrl:true*> ausgeführt, ansonsten werden die Schritte des Unterelements <*xrl:false*> ausgeführt.

- Durch das Element <*xrl:WHILE_DO*> werden Schleifen in den XRL+-Skripten definiert. Genauso, wie bei <*xrl:CONDITION*>, wird die Schleifenbedingung durch einen XPath-Ausdruck, der durch das Attribut *expression* beschrieben wird, festgelegt. Das Attribut *xml* referenziert das XML-Dokument.

- Das Element <*xrl:WAIT*> wird verwendet, um auf das Eintreffen von Ereignissen zu warten. Das Attribut *sync* spezifiziert die Anzahl der Ereignisse, die eintreffen sollen, bevor die Ausführung der Propagation fortgesetzt wird.

  Zwei Typen von Ereignissen können in der Propagationsbeschreibung spezifiziert werden: Ereignisse, die durch Nachrichten der Informationssysteme ausgelöst werden und Ereignisse durch den Timer ausgelöst werden.

Das Warten auf das Eintreffen von Nachrichten der Informationssysteme wird durch das Element <*xrl:MESSAGE_EVENT*> gesteuert. Die Attribute *system* und *schema* spezifizieren das Quellsystem und Quellschema der einzutreffenden XML-Nachricht. Diese wird durch das Attribut *xml_out* referenziert.

Das Element <*xrl:TIMER_EVENT*> realisiert ein Timeout während der Propagation. Durch das Attribut *time* wird die Dauer des Timeouts angegeben und das Attribut *type* bestimmt den zeitlichen Bezugspunkt.

- <*xrl:TERMINATE*> wird verwendet, um den Propagationsprozess vorzeitig abzubrechen.

Der Datenfluss im Propagationssystem wird durch die drei Operationen, die Transformation, das Filterung und die Propagation, definiert.

Eine Transformation wird durch das Element <*xrl:TRANSFORM*> eingeleitet. Die Attribute *xml_in* und *xml_out* sind die Bezeichner der Eingabe- und Ausgabedokumente. Die Transformation wird durch ein XSLT-Skript spezifiziert, der in einem Repository gespeichert ist. Das Attribut *xslt* referenziert dieses XSLT-Skript, der zur Durchführung der Transformation verwendet wird.

Durch das Element <*xrl:FILTER*> wird eine Filterungsoperation definiert. Es werden die Attribute *xml_in* und *xml_out* verwendet, um die Eingabe- und Ausgabedokumente der Operation zu definieren. Bei der Ausführung einer Filterungsoperation wird ein boolescher XPath-Ausdruck ausgewertet, indem eine Bedingung anhand des Eingabedokuments geprüft wird. Die Übergabe des XPath-Ausdrucks erfolgt durch das Attribut *expression*. Im Fall einer erfolgreichen Auswertung entspricht das Ausgabedokument dem Eingabedokument, andernfalls ist das Ausgabedokument leer.

Die Weiterleitung der transformierten Daten an die Zielsysteme wird durch das Element <xrl:PROPAGATE> spezifiziert. Die Attribute *system* und *schema* definieren das Zielsystem und Zielschema der Propagation, und das Attribut *xml* referenziert das weiterzuleitende XML-Dokument.

2.3 Die Architektur des SIES

Die Architektur des SIES ist in Abbildung 2.2 dargestellt. Das System besteht aus folgenden Hauptkomponenten: *dem Abhängigkeitsmanager*, *dem Propagationsmanager*, *dem Repository* und *den Adaptern*.

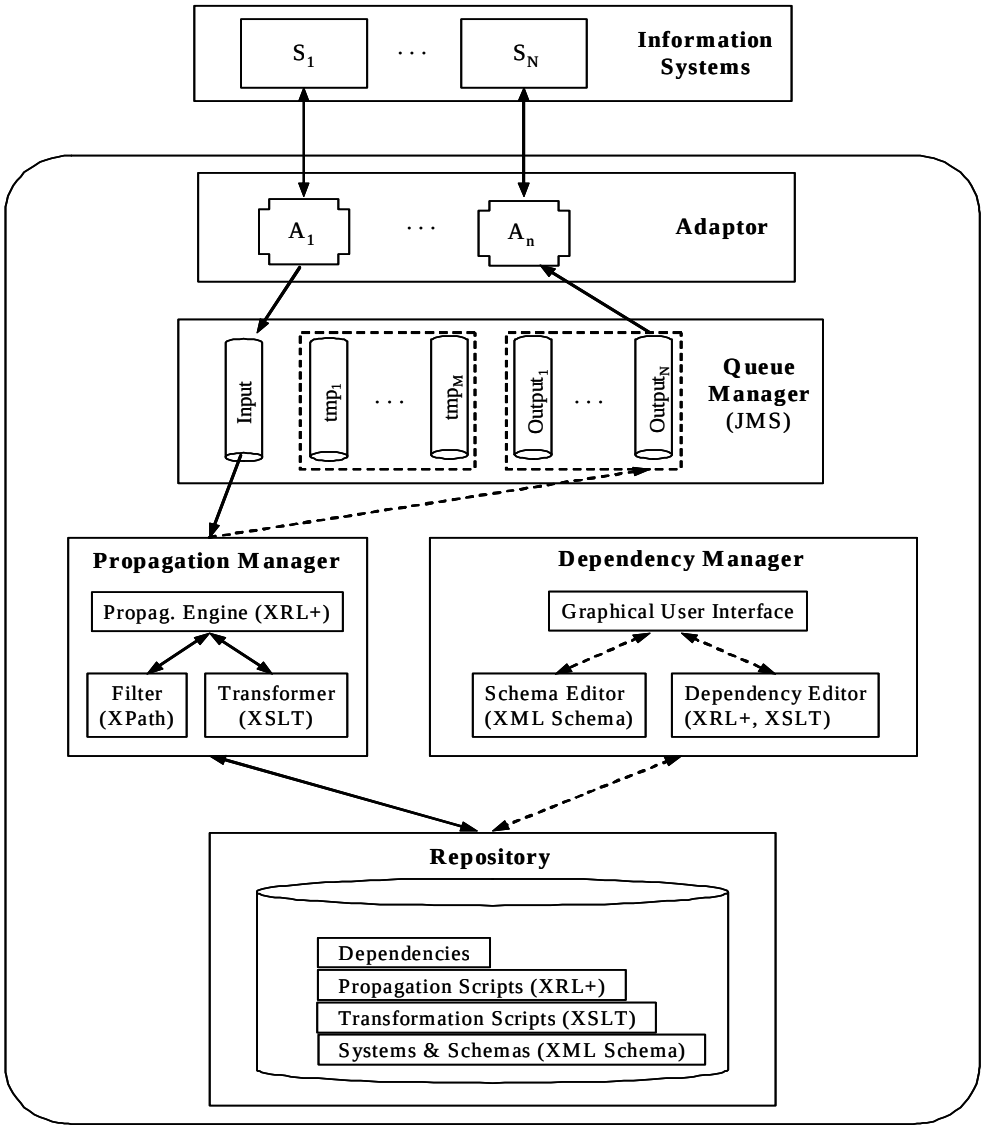

Abbildung 2.2: Die Architektur des SIES

2.3.1 Der Abhängigkeitsmanager

Der Abhängigkeitsmanager ist aus administrativen Gründen entstanden. Er hat die Aufgabe alle Informationen zu verwalten, die vom Propagationsmanager benötigt werden, um die Datenänderungen eines Quellsystems an die abhängigen Zielsysteme weiterzuleiten. In diesem Zusammenhang lassen sich zwei Funktionen des Abhängigkeitsmanagers herleiten. Die Erste besteht darin, die Datenrepräsentation jedes mit dem SIES verbundenen Informationssystems zu verwalten. Für diese Zwecke wurde *der Schema Editor* konzipiert. Der Schema Editor unterstützt die Erstellung, die

Änderung und das Löschen der unterschiedlichen Schemata jedes Informationssystems. Ein Schema beschreibt einen Teilaspekt eines Systems, z.B. eine Tabelle eines relationalen Datenbanksystems oder ein Objekt einer Java-Applikation.

Die zweite Funktion des Abhängigkeitsmanagers ist die Verwaltung der Datenabhängigkeiten zwischen den Quell- und Zielsystemen. Für diese Aufgabe wurde der Abhängigkeitseditor entwickelt. Der Abhängigkeitseditor ist eine interaktive Anwendung mit graphischer Oberfläche. Er bietet Editoren für die Abhängigkeits- (XRL+) und Transformationsskripte (XSLT). Diese Bestandteile der Datenabhängigkeiten werden im Repository gespeichert.

2.3.2 Der Propagationsmanager

Der Propagationsmanager ermöglicht die Transformation und Weiterleitung der Datenänderungen. Seine Komponenten sind der *Queue Manager*, der *Filter*, der *Transformer* und die *Propagationsengine*.

Der Queue Manager ist für die verlässliche Kommunikation zwischen dem Propagationsmanager und mit ihm verbundenen Informationssystemen verantwortlich. Er besteht aus einer Eingabe-Queue, die alle eingehenden Nachrichten empfängt, und mehreren Ausgabe-Queues und temporären Queues. Für jedes System wird eine eigene Ausgabe-Queue verwaltet. Die Weiterleitung der Datenänderungen für ein Informationssystem erfolgt durch seine eigene Queue. Die temporären Queues werden zur Speicherung von Zwischenergebnissen der Transformations- oder Filterungsoperationen verwendet. Die Realisierung des Queue Managers erfolgte mit Hilfe des Java Message Service (JMS) [SUN01].

Die Transformation der Daten im Propagationssystem erfolgt durch den Transformer. Diese Komponente des Propagationssystems wurde durch einen XSLT-Prozessor realisiert. Er bekommt als Eingabe XML-Nachrichten vom Queue Manager und XSLT-Skripte, zur Steuerung der Transformationsprozesse, vom Repository. Die Ergebnisse der Transformationen werden in die temporären Queues der Queue Manager zwischengespeichert und können entweder an die Zielsysteme weitergeleitet oder als Eingabe für weitere Transformationen benutzt werden.

Die Filterung im Propagationssystem wird durch den Filter umgesetzt. Genauso wie der Transformer bekommt er als Eingabe XML-Nachrichten vom Queue Manager. Er überprüft die Bedingung eines XPath-Ausdrücks anhand der XML-Daten. Auf diese Weise können die für die Weiterverarbeitung unnötigen Daten ignoriert werden.

Die Steuerung der Transformations- und Filterungsoperationen sowie die Weiterleitung der XML-Daten an die Zielsysteme ist Aufgabe der Propagationsengine. Zur Kommunikation zwischen der Propagationsengine und dem Queue Manager verläuft asynchron. Wenn eine Nachricht in die Eingabe-Queue des Queue Managers eintrifft, wird die Propagationsengine benachrichtigt. Sie holt die Nachricht ab, überprüft das Quellsystem und Quellschema der Nachricht und fragt alle Abhängigkeitsdefinitionen aus dem Repository ab, die diesem Quellsystem und Quellschema entsprechen. Das Ergebnis der Abfrage ist eine Menge von XRL+-Skripten. Die Propagationsengine interpretiert diese Skripten und startet die Transformationsoperationen auf dem

Transformer und die Filterungsoperationen auf dem Filter. Die Endergebnisse werden an die abhängigen Zielsysteme weitergeleitet.

2.3.3 Das Repository

Das Repository speichert alle Informationen, die für den Transformations- und Weiterleitungsprozess benötigt werden. Zu diesen Informationen zählen die Schemadefinitionen und Systemdefinitionen der Informationssysteme, die Abhängigkeitsskripte und Abhängigkeitsdefinitionen der Quell- und Zielsysteme und die Transformationsskripte. Weiterhin werden Informationen über die laufende Arbeit des Propagationssystems in Form eines Logging gesammelt. Diese Daten können benutzt werden, um Fehler im System zu beheben. Das Repository ist durch eine relationale Datenbank realisiert.

2.3.4 Adapter

Die Integration durch das Propagationssystem setzt voraus, dass die Eingabe- und Ausgabedaten im Queue Manager im XML-Format sind. Die einzelnen Informationssysteme sind autonom und verfügen in der Regel über keine XML-Kommunikationsschnittstellen. Aus diesem Grund wurden weitere Komponenten, die Adapter, eingeführt.

Ein Informationssystem wird durch einen eigenen Adapter an das Propagationssystem angeschlossen. Ein Adapter bildet die Daten eines Systems auf das entsprechende XML-Schema ab, das im Repository gespeichert ist. Umgekehrt werden die durch das Propagationssystem transformierten Daten durch die Adapter in das lokale Format der Informationssysteme umgewandelt. Auf diese Weise wird die Kommunikation zwischen dem Propagationssystem und den Informationssystemen in beide Richtungen ermöglicht, ohne dass Anpassungen an die Informationssysteme vorgenommen werden.

2.4 Processing Modell

In diesem Abschnitt werden die Ablaufschritte beschrieben, die der Propagationsmanager zur Weiterleitung einer Datenänderung durchführt. Einige Details zu diesem Thema wurden bereits im letzten Abschnitt erwähnt. Angesichts der Zielsetzung dieser Arbeit ist es notwendig, die Propagationsprozesse im Propagationssystem und einige technische Details zu erläutern.

2.4 Processing Modell

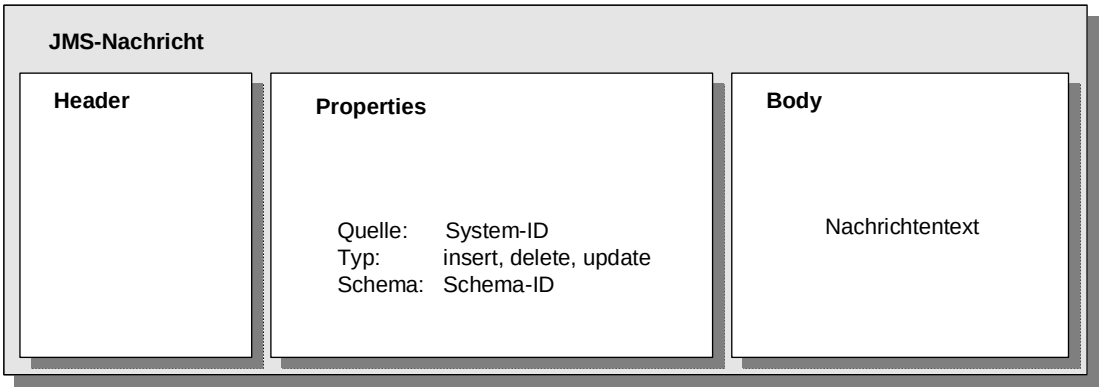

Abbildung 2.3: Aufbau einer JMS-Nachricht [HS02]

Die Integration der Informationssysteme erfolgt auf Basis der XML-Technologie. Die Kommunikation zwischen den Systemen und dem Propagationsmanager wird mit Hilfe des Java Message Service (JMS) umgesetzt. Die XML-Nachrichten, die die Datenänderungen des Quellsystems enthalten, werden durch den Propagationsmanager in *JMS-Nachrichten* eingepackt. Die Struktur einer JMS-Nachricht ist in Abbildung 2.3 dargestellt. Sie besteht aus einem *Header-*, einem *Property-* und einem *Body-Teil*. Die XML-Nachrichten werden in den Body-Teil der JMS-Nachrichten gespeichert.

Beim Start des Propagationssystems wird für die Eingabe Queue (*PMInput*) einen *Message Listener* registriert. Der Message Listener ist für die Übergabe von JMS-Nachrichten an die Propagationsengine verantwortlich. Wenn eine JMS-Nachricht in die Eingabe-Queue eintrifft, bekommt die Propagationsengine diese Nachricht durch den registrierten Message Listener. Die Namen des Quellsystems und des Quellschemas sind im Property-Teil der JMS-Nachricht gespeichert. Die Propagationsengine prüft zunächst im Repository, ob das Quellsystem und Quellschema existieren. Fall sie nicht existieren, wird die Nachricht nicht bearbeitet, ansonsten wird eine Abfrage an das Repository gesendet, um die definierten Abhängigkeiten für diese Nachricht zu bekommen (siehe Abbildung 2.4). Das Ergebnis der Abfrage sind alle Propagationsskripte (XRL+-Skripte), die ausgeführt werden müssen, um die in der XML-Nachricht enthaltene Datenänderungen an alle abhängigen Zielsysteme weiterzuleiten.

Als Nächstes werden die XRL+-Skripte nacheinander geparst. Eine interne Datenstruktur wird für jedes geparste Skript aufgebaut. Schließlich wird für jedes Propagationsskript eine eigene *Prozessinstanz* mit der erzeugten Datenstruktur initialisiert und gestartet. Eine Prozessinstanz im Propagationssystem ist verantwortlich für die Ausführung einer Abhängigkeitsdefinition und läuft in ihrem eigenen Thread ab. Die Propagationsengine kann mit der Bearbeitung der nächsten Nachricht aus der Eingabe Queue beginnen, sobald die Prozessinstanz für die aktuelle Nachricht initialisiert und gestartet ist.

Wie in Abschnitt 2.2 beschrieben wurde, kann ein Propagationsskript mit Hilfe des Elements *<xrl:PARALLEL>* eine parallele Ausführung von Tasks definieren. Dieses Konzept wird realisiert, indem für jedes Unterelement ein neuer Thread gestartet wird. Falls mehrere Prozessoren auf einem Rechensystem zur Verfügung stehen, werden diese Unterelemente parallel ausgeführt.

Anschließend können folgende Punkte zusammengefasst werden:

- Die Initialisierung der Prozessinstanzen durch die Propagationsengine verläuft sequenziell.
- Die Prozessinstanzen werden in eigenen Threads unabhängig voneinander ausgeführt.
- Die Elemente einer Prozessinstanz können parallel ausgeführt werden, falls sie in der Prozessbeschreibung (XRL+-Skript) als parallel definiert sind.

In Abbildung 2.4 ist ein Beispiel für die Weiterleitung einer Datenänderung an zwei Zielsysteme dargestellt. Der Propagationsprozess besteht aus zwei parallelen Transformations- und Propagationsvorgängen.

Anhand dieses Beispiels werden im Folgenden einige Definitionen eingeführt. Das Eintreffen der Nachricht in den Queue Manager erfolgt im Zeitpunkt (1) und das Empfangen der Nachricht durch die Propagationsengine im Zeitpunkt (2). Das Zeitintervall zwischen diesen zwei Ereignissen ist die Wartezeit der Nachricht im Queue Manager. Da die Übermittlung der Nachrichten zwischen dem Queue Manager und der Propagationsengine asynchron verläuft, können keine Zeitschranken für die Ausführung dieser Operation gesetzt werden.

Im Zeitpunkt (3) wird die Prozessinstanz gestartet. Das Zeitintervall zwischen Zeitpunkt (2) und Zeitpunkt (3) wird als *Prozessinitialisierung* bezeichnet.

Unter *Prozessausführungszeit* wird die Zeit zwischen dem Prozessstart (Zeitpunkt (3)) und dem Prozessende (Zeitpunkt (4)) verstanden.

Die Summe aus der Prozessinizialisierungszeit und der Prozessausführungszeit ergibt die *Bearbeitungszeit* einer Nachricht durch den Propagationsmanager. In der Bearbeitungszeit ist die Wartezeit einer Nachricht im Queue Manager nicht enthalten. Aufgrund der gewählten asynchronen Kommunikation, muss zwischen der Bearbeitungszeit durch den Propagationsmanager und der *Propagationszeit* aus der Sicht der Informationssysteme, die durch das Zeitintervall zwischen Zeitpunkt (1) und Zeitpunkt (5) definiert ist, unterschieden werden.

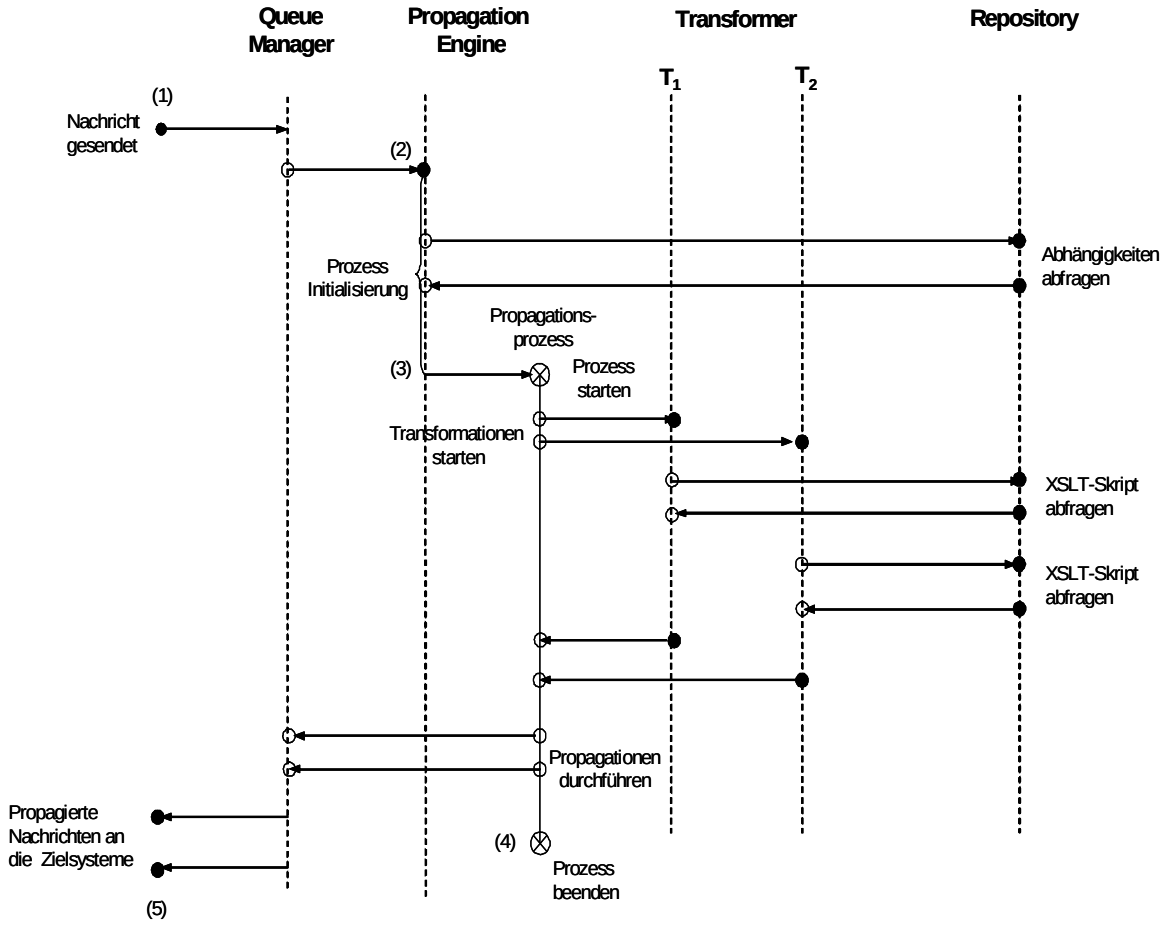

Abbildung 2.4: Processing Modell

2.5 Die Leistung des Propagationssystems

2.5.1 Der Leistungsbegriff

In den vorangegangenen Abschnitten wurde der Leistungsbegriff mehrmals erwähnt, ohne dass dabei eine Definition für diesen Begriff gegeben wurde. In der Literatur (vor allem in der englischsprachigen) wird der Begriff Leistung (engl. Performance) ganz allgemein verwendet, um die Funktionen eines Systems qualitativ oder quantitativ zu charakterisieren. Leistung in dieser breiten Interpretation kann folgende Eigenschaften eines Systems zusammenfassen [CG01]:

- Zum einen definiert der Begriff *die Leistungskraft des Systems*, charakterisiert durch die Zeit, die es benötigt, um ein Problem zu lösen, oder durch die Anzahl der gefundenen Lösungen für ein gegebenes Zeitintervall.

- Eine zweite sehr häufig benutzte Interpretation dieses Begriffs drückt *die Verfügbarkeit* eines Systems aus, die durch die Größe „time to failure“ charakterisiert wird.

- Des Weiteren gibt es noch eine dritte Interpretation, die *die Korrektheit* der durch ein System gelieferten Lösung und *die ergonomischen Eigenschaften* des Systems kennzeichnet.

In dieser Arbeit wird unter dem Begriff „Leistung“ ausschließlich die erste Interpretation verstanden.

2.5.2 Leistungsfaktoren

Im Allgemeinen gibt es drei Faktoren, die die Leistung jeder Applikation beeinflussen:
- die Hardwareressourcen,
- die Software,
- und die Arbeitslast.

Die Hardwareressourcen sind z.B. die zur Verfügung stehenden CPUs, der Haupt- und der Hintergrundspeicher. Der zweite Faktor ist die Software. Bezogen auf das Propagationssystem zählen dazu das Betriebsystem, das Middleware (JMS), die verwendete Datenbank für das Repository, die Java-Umgebung zur Ausführung des Propagationssystems.

Die Hardware- und Softwareressourcen definieren die Arbeitsumgebung (die Systemkonfiguration) des Propagationsmanagers. Zur Untersuchung der Leistung des Propagationssystems wird seine Arbeitsumgebung in dieser Arbeit als konstant betrachtet.

Die Arbeitslast eines Systems ist die Menge aller Aufträge und Daten, die während einer Beobachtungsperiode bearbeitet werden [LH92]. Die Arbeitslast des Propagationssystems wird durch die verbundenen Informationssysteme erzeugt. Sie senden ihre Datenänderungen und lösen dadurch unterschiedliche Propagationsprozesse aus, die dann Transformations- und Filterungsoperationen durchführen.

Die Arbeitslast, die auf das Propagationssystem angewendet wird, lässt sich durch folgende Daten beschreiben:

- **Die Datenänderungsnachrichten** lösen die Propagationsprozesse im Propagationssystem aus. Die Größe und die Komplexität der Datenänderungen beeinflussen die Arbeit des Transformers, des Filters und des Queue Managers.

- **Die auszuführenden XRL+-Skripte** bilden die zweite Komponente der Arbeitslast, indem sie die Anzahl und die Reihenfolge der Transformations-, Filterungs- und Propagationsvorgänge definieren.

- **Die Menge der XSLT-Skripte** spezifiziert die Transformationen innerhalb der Propagationsprozesse. Die Komplexität der Transformationsskripte ist von entscheidender Bedeutung für die Bearbeitungszeit der Datenänderungen und den Durchsatz des Propagationssystems.

- Der letzte Faktor für die Leistung des Propagationssystems ist **die zeitliche Charakteristik der Arbeitslast**. Zur Beschreibung der zeitlichen Charakteristik der Last wird eine Funktion benötigt, die das Eintreffen von Datenänderungen

definiert. Wenn mehrere Datenänderungen gleichzeitig gesendet werden, werden im Propagationssystem mehrere Propagationsprozesse ausgelöst, die dann parallel ablaufen. Wenn diese Datenänderungen stattdessen in regelmäßigen Zeitabständen gesendet werden, so dass sie sequenziell abgearbeitet werden, sind im Propagationsmanager weniger Prozesse aktiv und weniger Systemressourcen werden auf einmal gebunden.

Die Mengen der XRL+- und der XSLT-Skripte sind endlich. Sie sind im Repository gespeichert und werden in der Regel selten geändert. Die Menge der XML-Nachrichten, die die Propagationsprozesse auslösen, ist dagegen unendlich.

In den nächsten Kapiteln werden die Begriffe *statische* und *dynamische Charakteristik* in Bezug auf die Arbeitslast und auf das Propagationssystem verwendet. Unter statischer Charakteristik der Last wird das Senden einzelner Datenänderungen verstanden. Die Leistung des Propagationssystems unter dieser Last wird mit dem Begriff statische Charakteristik des Propagationssystems bezeichnet. Werden gleichzeitig mehrere Nachrichten durch den Propagationsmanager abgearbeitet, dann spricht man von dynamischer Charakteristik der Leistung des Propagationsmanagers und der Arbeitslast.

2.5.3 Offene Probleme bei der Realisierung des Propagationsmanagers

Das Konzept des SIES wird ständig weiterentwickelt und an die aktuellen Probleme des Teilprojekts A5 angepasst. Bei der Realisierung des Konzepts gibt es noch offene Fragen, die zurzeit diskutiert werden. Da einige von diesen Problemen einen direkten Einfluss auf die Leistung des Propagationssystems haben, werden sie hier kurz erwähnt.

2.5.3.1 Nachrichtenübergabe an die Prozesseinstanzen

Wenn eine Nachricht in die Eingabe-Queue eintrifft, wird sie durch den registrierten Message Listener an die Propagationsengine übergeben. Ein oder mehrere Prozessinstanzen werden dann gestartet. Die ursprüngliche Nachricht ist dann nicht mehr in der Eingabe-Queue. Deswegen gibt es zur Übergabe der Nachrichten an die Prozessinstanzen zurzeit zwei Lösungen:

- Übergabe durch erneutes Senden der Nachricht an die Eingabe-Queue: Sobald eine Prozessinstanz gestartet ist, registriert er bei der Eingabe-Queue seinen eigenen Message Listener. Die Propagationsengine wartet nach dem Prozessstart ein Zeitintervall (konfigurierbar) und sendet die ursprüngliche Nachricht erneut an die Eingabe-Queue. Die Prozessinstanz kann sie dann von der Eingabe-Queue holen.

- Direkte Übergabe: Die Nachricht wird im Hauptspeicher gespeichert und als Parameter übergeben, sobald die Prozessinstanz gestartet ist.

Die Übergabe durch erneutes Senden verursacht lange Bearbeitungszeiten und basiert auf die Annahme, dass nach einem Zeitintervall die gestartete Prozessinstanz seinen Message Listener schon registriert hat. Die direkte Nachrichtenübergabe ist dagegen

schneller, kann aber beim Absturz des Propagationsmanagers zur Verlust von Nachrichten führen.

Da beide Lösungen nicht zufrieden stellend sind, werden zurzeit zwei weitere Lösungen für dieses Problem diskutiert. Beide basieren zum Teil auf den ersten Lösungsansatz. Es wird allerdings nicht die Eingabe Queue sondern eine weitere Warteschlange zur Übermittlung der Nachrichten von der Propagationsengine an die Prozessinstanzen verwendet.

2.5.3.2 Das Reihenfolgeproblem

Das Reihenfolgeproblem tritt auf, wenn zwei Nachrichten, die durch das gleiche System und Schema gesendet sind, parallel ausgeführt werden. In diesem Fall kann es passieren, dass die zweite Nachricht schneller bearbeitet wird als die erste und die Zielsysteme die propagierten Nachrichten in umgekehrter Reihenfolge erhalten. Das Ergebnis dieser beiden Datenänderungenspropagationen kann zur Inkonsistenzen der Zielsystemdaten führen, wenn zum Beispiel die erste Nachricht ein „insert“ und die zweite ein „delete“ auf die gleiche Datenmenge durchführt.

2.5.3.3 Das Bestätigungsproblem

Wenn eine Prozessinstanz ausgeführt ist, wird die ursprüngliche Nachricht bestätigt, damit sie von der Eingabe-Queue entfernt werden kann. Das Bestätigungsproblem tritt auf, wenn bei paralleler Ausführung die Nachrichten in umgekehrter Reihenfolge fertig werden. Dann wird zuerst die zweite Nachricht bestätigt, was automatisch laut JMS-Spezifikation [SUN01] zur Bestätigung der ersten Nachricht führt. Bei einem Absturz des Propagationssystems gehen dann Daten verloren.

Der aktuelle Stand des Propagationssystems definiert zurzeit keine Maßnahmen, um das Reihenfolge- und das Bestätigungsproblem zu lösen. Eventuelle Maßnahmen zur Lösung dieser Probleme würden dazu führen, dass Nachrichten für längere Zeiten im Propagationsmanager oder in den temporären Queues des Queues Managers zwischengespeichert werden. Demzufolge würden sich diese Maßnahmen auf die Bearbeitungszeit und den Durchsatz auswirken.

Kapitel 3

Leistungserfassung des Propagationssystems

Dieses Kapitel beschäftigt sich mit der Leistungserfassung des Propagationsmanagers. Zunächst wird kurz auf das Thema Leistungsmessung und Leistungsbewertung eingegangen. Die in der Literatur existieren Methoden zur Leistungsbewertung von Computer-Systemen werden vorgestellt. Als Nächstes wird ein Konzept zur Leistungserfassung des Propagationssystems entwickelt und schließlich werden die Anforderungen an das zu entwickelnde Tool zur Leistungserfassung definiert.

3.1 Methoden zur Leistungsbewertung

Einen Überblick über die Methoden zur Leistungsbewertung eines Systems [LH92] gibt Abbildung 3.1.

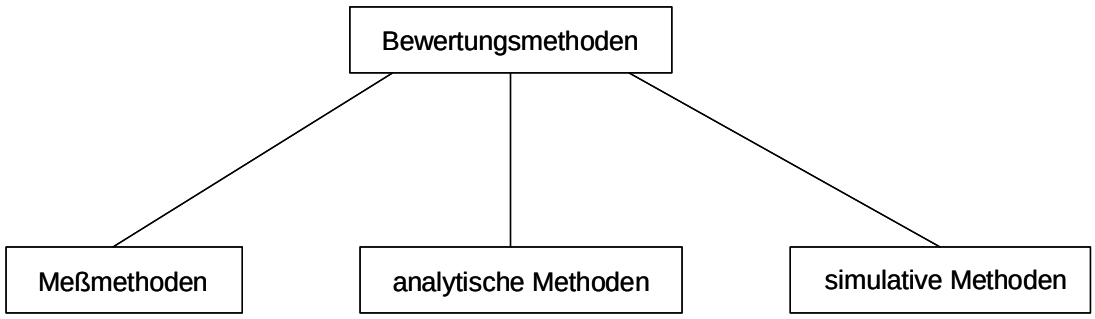

Abbildung 3.1: Methoden zur Leistungsbewertung

3.1.1 Meßmethoden

Meßmethoden werden angewendet, wenn das zu untersuchende System zur Verfügung steht. Zur Leistungsmessung eines Systems mit einer bestimmten Arbeitslast werden so genannte Benchmark-Tests durchgeführt und interessante Größen gemessen. Durch die Analyse der gemessenen Daten können potenzielle Engpässe gefunden und durch entsprechende Verbesserungsmaßnahmen beseitigt werden.

3.1.2 Analytische Methoden

Falls das System nicht zur Verfügung steht oder Untersuchungen mit dem realen System zu aufwendig sind, werden *analytische Methoden* verwendet. Hierbei wird ein Modell für das System aufgebaut und mit seiner Hilfe die Leistung des Systems untersucht. Modelle werden zum einen in der Anfangsphase der Systementwicklung verwendet, da zu diesem Zeitpunkt Messungen nicht möglich sind. Zum anderen können alternative Systemkonfigurationen miteinander verglichen werden. Weiterhin kann das Verständnis über die Systemleistung durch ein Modell verdeutlicht werden.
Für die analytischen Methoden bieten sich unterschiedliche Modellierungstechniken an, wie z.B. die Operationale Analyse [LH92], die Theorie der Warteschlangen [LH92][BB86], die Timed-Petri-Netzen [BA90] usw.

Dieser Typ von Bewertungsmethoden hat auch einige Nachteile. Vor allem können wichtige Systemeigenschaften bei der Modellbildung vernachlässigt werden oder unwichtige Systemeigenschaften können im Modell enthalten sein. Demzufolge ist das entstandene Leistungsmodell keine gültige Darstellung der Wirklichkeit und das Systemverhalten wird nicht richtig dargestellt.

3.1.3 Simulative Methoden

Bei den *simulativen Methoden* wird das dynamische Verhalten eines Systemmodells experimentell untersucht. Die simulativen Methoden liefern Ergebnisse für Modelle, die wegen ihrer Komplexität nicht erfassbar sind, oder für die keine analytischen Lösungen existieren.

3.2 Leistungserfassung des Propagationsmanagers

Da das Propagationssystem bereits als Prototyp zur Verfügung steht, wird es anhand unterschiedlicher Messungen analysiert. Zunächst wird das Modell der Testumgebung vorgestellt, mit deren Hilfe die Durchführung der Messungen ermöglicht wird. Als Nächstes werden die durchzuführenden Messszenarien diskutiert. Die konkreten Testfälle für die Leistungserfassung des Propagationsmanagers werden in Kapitel 5 vorgestellt.

3.2.1 Abstraktes Modell der Testumgebung

Die Testumgebung besteht, wie in Abbildung 3.2 dargestellt ist, aus *einem Lastgenerierungstool* und dem Propagationsmanager. Die Aufgabe des Lastgenerierungstools besteht hauptsächlich darin, Arbeitslast für den Propagationsmanager durch Senden von Nachrichten zu erzeugen. Diese Nachrichten sollen Datenänderungen einzelner Quellsysteme simulieren.

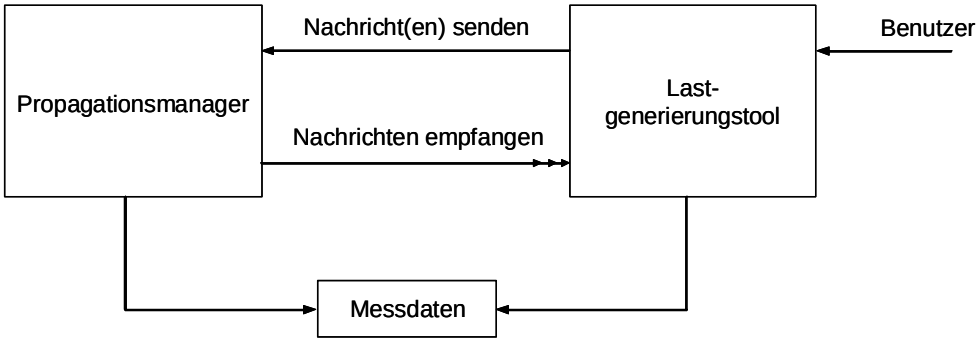

Abbildung 3.2: abstraktes Modell der Testumgebung

Die Arbeit des Lastgenerierungstools wird durch den Benutzer gesteuert und läuft in mehrere Schritte ab. Der Benutzer gibt einen Befehl ein, das Lastgenerierungstool erkennt den Befehl und sendet eine oder mehrere Nachrichten an den Propagationsmanager. Der Propagationsmanager bearbeitet die eingetroffenen Nachrichten und sendet die transformierten Daten an das Lastgenerierungstool zurück. Auf diese Weise kann die Arbeit des Propagationsmanagers simuliert werden. Neben der Bearbeitung der eingetroffenen Nachrichten werden Daten für die Leistung des Propagationsmanagers ermittelt und gespeichert werden. Nach der Beendigung der Messexperimente können die Messdaten analysiert werden.

3.2.2 Messszenarien

Zur Analyse der Arbeit des Propagationssystems können zwei Typen von Messszenarien durchgeführt werden:

- Die Erfassung der statischen Charakteristik erfolgt durch Senden einzelner Nachrichten an den Propagationsmanager. Hierbei werden unterschiedliche Größen gemessen, wie z.B. die Bearbeitungszeit der eingetroffenen Nachrichten (siehe Abschnitt 3.3.1).

- Beim zweiten Typ von Messszenarien wird das dynamische Verhalten des Propagationssystems erfasst. Das Ziel dieser Messungen ist die Arbeit des Systems auf mögliches Fehlverhalten zu untersuchen und mögliche Engpässe im System zu finden. Die Messgrößen, die zur Untersuchung der statischen Charakteristik verwendet werden, können bei diesen Messszenarien sehr eingeschränkt eingesetzt werden, da die Informationen, die sie liefern, nicht ausreichend sind, um die Engpässe des Systems zu analysieren (siehe Abschnitt 3.3.2).
  Das Senden von Nachrichten durch das Lastgenerierungstool an den Propagationsmanager kann in regelmäßigen und unregelmäßigen (zufälligen) Zeitabständen erfolgen. Für die Ziele dieser Arbeit wird nur der erste Fall erforscht, da zum einen die Wiederholbarkeit der Experimente garantiert wird. Zum anderen ist es nicht ersichtlich, ob durch das Senden von Nachrichten in zufälligen Zeitabständen zusätzliche Erkenntnisse über die Leistung des Propagationsmanagers gewonnen werden können.

3.3 Messgrößen

In den unterschiedlichen Veröffentlichungen und Arbeiten über den Propagationsmanager wurden die funktionalen und nicht-funktionalen Anforderungen an den Propagationsmanager definiert. Als Nächstes erfolgte eine prototypische Implementierung des Propagationsmanagers. Weiterhin findet man in [MB03] ein typisches Einsatzszenario für den Propagationsmanager zur Ankopplung des Auftragsmanagement-Prüfstands (AMP) an das Fabriklayout-Planung-System (FLP). Zur Leistungsbewertung des Propagationssystems sind bis jetzt keine Richtlinien ausgearbeitet worden, was ein wichtiger Schritt zur Verbesserung des Systems darstellt.

Eine Zusammenfassung von Messgrößen zur Bewertung von Computer-Systemen ist in Tabelle 3.1 gegeben.

Leistungsmaße	Definition
Durchsatz	Anzahl von Aufträgen einer Arbeitslast, die pro Zeitintervall bearbeitet werden.
Kapazität	Maximaler Durchsatz in einem Zeitintervall
Bearbeitungszeit	Gesamtzeit, die ein Auftrag vom System bearbeitet wird.
Wartezeit	Gesamtdauer, die ein Auftrag auf seine Bedienphasen wartet.
Antwortzeit	Aufenthaltszeit eines Auftrags im System, bis zum Eintreffen des Resultats an einer Ausgabeschnittstelle.
Auslastung	Prozentsatz der Zeit, den eine Systemkomponente produktive Arbeit leistet.

Tabelle 3.1: Verwendete Leistungsmaße[LH02]

Zwei Größen können zur Beurteilung der Leistung des Propagationsmanagers verwendet werden:

- die Antwortzeit (engl. response time) einer Datenänderungspropagation und
- der Durchsatz des Propagationsmanagers.

Die Antwortzeit einer Datenänderungspropagation gibt Hinweise darauf, wie effektiv einzelne Datenänderungen an die Zielsysteme weitergeleitet werden. Der Durchsatz des Propagationssystems wird definiert durch die Anzahl der durchgeführten Propagationen innerhalb eines Zeitraums.

An dieser Stelle sollte man die Bearbeitungszeit von der Antwortzeit einer Datenänderungspropagation abgrenzen. Die Antwortzeit einer Datenänderungs-propagation schließt neben der Bearbeitungszeit und der Wartezeit einer Nachricht in die Eingabe-Queue auch die zwischen dem Queue Manager und dem Propagationsmanager angefallenen Kommunikationskosten ein. Hierbei handelt es sich um eine asynchrone Übermittlung der Datenänderungen. Die Semantik asynchroner Kommunikation definiert keinen zeitlichen Rahmen für diese Übermittlung. Im Allgemeinen kann die Nachrichtenübergabe zwischen dem Queue Manager und dem Propagationsmanager zu jedem beliebigen Zeitpunkt stattfinden. Um die Leistung des Propagationsmanagers zu steigern wird deshalb die Bearbeitungszeit näher untersucht.

3.3.1 Bestandteile der Bearbeitungszeit

Die Bearbeitungszeit ist ein wesentliches Kriterium für die Leistung des Propagationssystems. Man kann damit die am System durchgeführten Optimierungen bewerten und Aussagen machen, ob sie zur Leistungssteigerung geführt haben. Leider stellt die Bearbeitungszeit keine Entscheidungsgrundlage für die notwendigen Verbesserungen am System dar. Somit ist diese Größe nicht geeignet, um Entscheidungen bezüglich Systemverbesserungen zu treffen. Zusätzliche Messgrößen und Messungen sind aus diesem Grund notwendig.

In Kapitel 2 wurde bereits die Bearbeitungszeit in folgende Bestandteile unterteilt:

- Initialisierungszeit,
- Ausführungszeit und
- Beendigungszeit einer Prozessinstanz.

Neben dieser Aufteilung der Bearbeitungszeit werden noch folgende Messgrößen betrachtet:

- Die *Repository-Abfragezeiten* einer Prozessinstanz geben Informationen darüber wie viel Zeit eine Prozessinstanz wartet, um Daten, die im Repository gespeichert sind, zu bekommen. Dazu gehören die Abfragezeit für die XRL+- und XSLT-Dokumente und die Abfragezeit für die Auflösung der Datenabhängigkeiten. Weiterhin werden Log-Einträge ins Repository gespeichert.

- Unter der *Transformationszeit* wird die Zeit verstanden, die den Transformer benötigt, um eine XML-Datei in eine andere XML-Datei mit Hilfe eines XSLT-Dokuments umzuwandeln. In dieser Größe ist die Repository-Abfragezeit für das entsprechende XSLT-Dokument nicht enthalten.

- Als *Filterungszeit* wird das Zeitintervall bezeichnet, die der Filter benötigt, um eine in der Prozessbeschreibung definierte Filterungsoperation auszuführen.

- Ein weiterer Bestandteil der Bearbeitungszeit ist die *Propagationszeit*. Darunter wird die vom Propagationsmanager benötigte Zeit, eine in der Prozessbeschreibung definierten Propagationsoperation auszuführen, verstanden.

- Das Parsen der XRL+-Dokumente sollte auch untersucht werden. Vor allem ist die Abhängigkeit zwischen der Größe der XRL+-Dokumente und dem *Parsenaufwand* interessant.

Diese Aufzählung ist nur ein Versuch die Bestandteile der Bearbeitungszeit zu systematisieren. Im Laufe der Arbeit können sich auch andere, nicht durch diese Messgrößen erfasste, Vorgänge ergeben, die einen großen Einfluss auf die Leistung des Propagationsmanagers haben.

3.3.2 Erfassung des dynamischen Verhaltens

Die Erfassung der Bearbeitungszeit und deren Bestandteile kann verwendet werden, um einzelne Datenänderungspropagationen zu untersuchen. Eine Analyse der dynamischen Charakteristik des Propagationssystems anhand dieser Messgrößen erweist sich aber als sehr schwierig. Die unübersichtlichen Messdaten erschweren das Auffinden von Engpasskomponenten des Propagationssystems. Als Engpasskomponenten werden hier die Komponenten bezeichnet, die den größten Einfluss auf die Leistung des Gesamtsystems haben. Zur Erfassung des dynamischen Verhaltens des Propagationsmanagers müssen andere Messgrößen verwendet werden.

Zur Analyse der Leistung der einzelnen Komponenten wird auf Messgrößen zurückgegriffen, die in der Operationalen Analyse [LH92][DP91] verwendet werden. Man betrachte die Arbeit einer einzelnen Komponente des Systems, wie in Abbildung 3.3 dargestellt ist. Zur Erfassung der Leistung der Komponente werden ein Zähler und eine Uhr verwendet. Der Zähler (counter B) zählt die Anzahl der durch die Komponente abgearbeiteten Aufträge (ein Auftrag kann z.B. für das Repository eine Datenbankanfrage und für den Transformer die Durchführung einer Transformation sein) einer Komponente. Die Uhr (in Abbildung 3.3 als *timer A* bezeichnet) erfasst die Zeit, die eine Komponente benötigt, um die Aufträge zu bearbeiten. Mit Hilfe dieser Bezeichnungen können folgende Größen definiert werden:

- Die Auslastung der Komponente: $U = \frac{A}{T}$, wobei T die Gesamtzeit zur Durchführung des Messexperiments ist. Aus dieser Definition folgt, dass die Auslastung immer im Intervall [0,1] liegt. Durch die Analyse der Auslastungen der einzelnen Komponenten des Systems können Schlussfolgerungen über mögliche Engpässe im Propagationssystem gezogen werden. Vor allem die Komponenten mit einem hohen Auslastungswert weisen auf einen möglichen Engpass hin.

- Der Durchsatz der Komponente ist: $D = \frac{B}{T}$.

- Der Kapazität der Komponente ist: $K = \frac{BT}{A}$. Der Kapazität einer Komponente gibt an, wie viele Aufträge durch die Komponente in einem Zeitintervall abgearbeitet werden können (Maximaler Durchsatz in einem Zeitintervall).

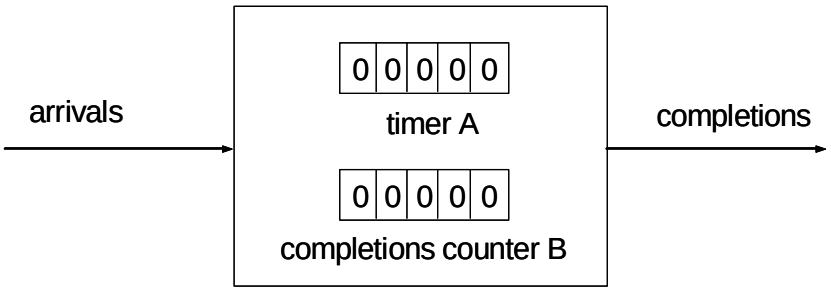

Abbildung 3.3: Analyse einzelner Komponenten des Propagationsmanagers

Bei der Erfassung des dynamischen Verhaltens des Propagationssystems werden folgende Komponenten betrachtet (siehe Abbildung 3.4):

- Die Propagationsengine, die für die Initialisierung der einzelnen Prozessinstanzen verantwortlich ist.
- Der Transformer und der Filter werden als eine Komponente betrachtet, da sie durch dieselbe Klasse umgesetzt sind.
- Der Queue Manager, der für die Durchführung der Propagate-Operationen verantwortlich ist.
- Die letzte Komponente, die bei der Erfassung des dynamischen Verhaltens des Propagationssystems beobachtet wird, ist das Repository.

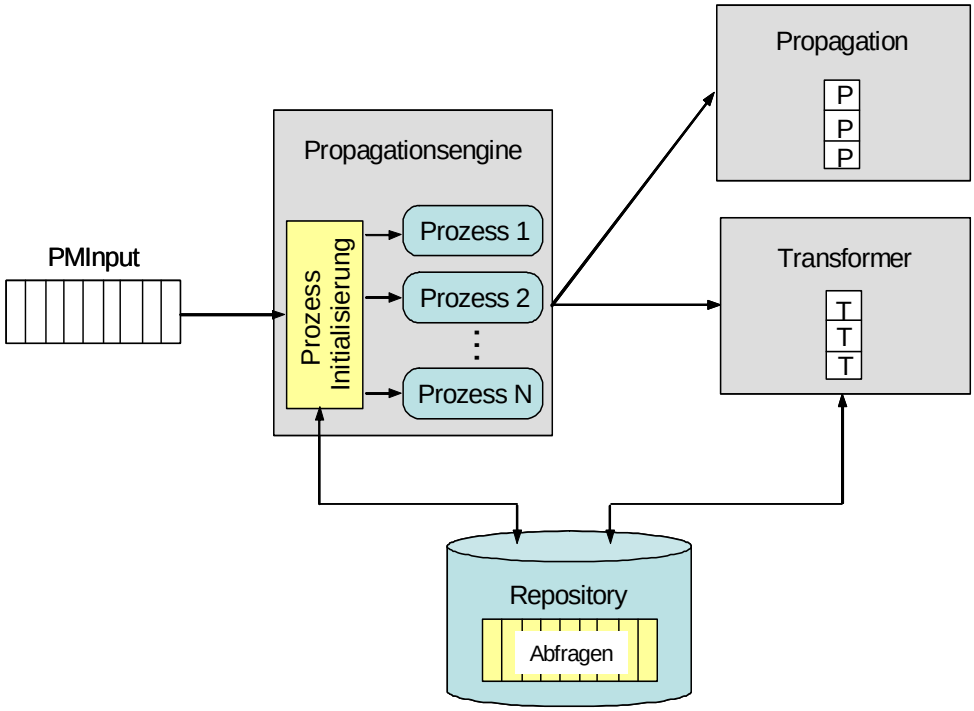

Abbildung 3.4: Dynamisches Modell der Leistungserfassung

3.4 Anforderungen an die Testumgebung

Im Folgenden werden die Anforderungen an das zu entwickelnde Lastgenerierungstool beschrieben. Weiterhin werden die Anforderungen an die Erweiterungen am Propagationsmanager definiert, die zur Leistungserhebung des Systems notwendig sind.

3.4.1 Abhängigkeitstypen

Die Spezifikation der Transformationsprozesse wird mit Hilfe der XRL+- und XSLT-Dokumente, die im Repository gespeichert sind, gesteuert. Der Typ der Abhängigkeitsdefinitionen hat, wie es im Folgenden gezeigt wird, eine direkte Auswirkung auf die Funktionalität des Lastgenerierungstools.

Die Abhängigkeiten zwischen den Quell- und Zielsystemen im SIES können in vier verschiedene Klassen unterteilt werden:

- Eine *1:1*-Abhängigkeit besteht genau dann, wenn zwei Systeme beteiligt sind und zwar ist das erste System ein Quellsystem und das zweite ein Zielsystem.

- Bei einer *1:n*-Abhängigkeit gibt es ebenfalls ein Quellsystem, dessen transformierten Datenänderungen jedoch an mehrere Zielsysteme weitergeleitet werden.

- Falls ein Zielsystem an den gleichzeitigen Datenänderungen in *n* verschiedenen Systemen interessiert ist, dann besteht zwischen diesen Systemen eine *n:1*-Abhängigkeit. Gleichzeitig bedeutet in diesem Fall, dass alle Datenänderungen innerhalb eines festgelegten Zeitraums stattfinden.

- Bei einer *n:m*-Abhängigkeit gibt es *m* Zielsysteme, die an den gleichzeitigen Änderungen von Daten in *n* Quellsystemen interessiert sind.

Bei der Untersuchung der Leistung des Propagationssystems mit Hilfe der Testumgebung, werden ausschließlich *1:1*- und *1:n*-Abhängigkeiten betrachtet, da normalerweise diese zwei Typen von Abhängigkeiten zwischen Systemen bestehen. Diese Einschränkung bedeutet für das dargestellte Modell der Testumgebung, dass Datenänderungspropagationen durch einzelne Nachrichten ausgelöst werden können.
Die anderen zwei Abhängigkeitstypen werden weiterhin im Rahmen des Teilprojekts A5 untersucht. Es ist im Rahmen dieses Projekts derzeit nicht ersichtlich, ob es notwendig und praktikabel ist, auf gleichzeitige Datenänderungen zu reagieren.

Neben dieser Klassifizierung der Abhängigkeitstypen, kann noch eine weitere Unterteilung der *1:1*- und *1:n*-Abhängigkeiten in Betracht gezogen werden, die für die Funktionalität des Lastgenerierungstools wichtig ist. Das Problem wird anhand Abbildung 3.5 erklärt. Es handelt sich in diesem Beispiel um eine zyklische Datenabhängigkeit zwischen dem AMP-Prüfstand und dem FLP-System. Beide Systeme sind sowohl Quell- als auch Zielsysteme. Damit diese Abhängigkeit realisiert werden kann, muss die entsprechende Prozessinstanz zwischendurch auf Nachrichten der Systeme warten.
Das Lastgenerierungstool sollte in der Lage sein auf die empfangenen transformierten Nachrichten zu reagieren und dementsprechend neue Nachrichten an das Propagationssystem zu senden, damit Abhängigkeiten, wie diese zwischen dem AMP-Prüfstand und dem FLP-System, durch das Lastgenerierungstool simuliert werden können. Ähnlicher Ansatz ist das Benchmarking von Workflow-Systemen. In [GM02] ist ein Benchmark für Workflow-Systeme vorgeschlagen, der auf dem TPC-C [TPC] Benchmark basiert.
Um diesen Ansatz zu realisieren, muss neben der Definition der Abhängigkeiten zwischen den verbundenen Informationssystemen zusätzlich die Funktionalität der

einzelnen Systeme implementiert werden. Es ist außerdem nicht ersichtlich, ob die Simulation solcher Datenabhängigkeiten entscheidende Erkenntnisse zur Verbesserung des Propagationssystems bringen würden. Aus diesen Gründen wird dieser Typ von *1:n*-Datenabhängigkeiten im Rahmen dieser Arbeit nicht betrachtet.

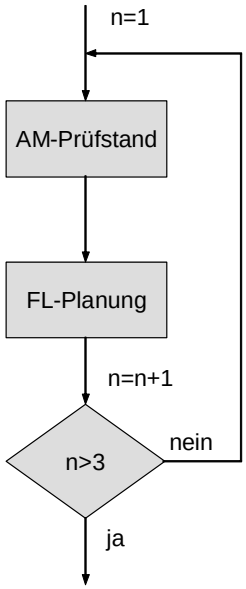

Abbildung 3.5: Datenfluss bei iterativer Ausführung von AMP- und FLP-System [KO00]

3.4.2 Funktionale Anforderungen an das Lastgenerierungstool

3.4.2.1 Kopplung zwischen dem Propagationsmanager und dem Lastgenerierungstool

Das Lastgenerierungstool soll unabhängig vom Propagationsmanager realisiert werden, so dass es auch in einer verteilten Umgebung verwendet werden kann. Um diese Unabhängigkeit zu gewährleisten, sollte die Kommunikation zwischen dem Propagationsmanager und dem Lastgenerierungstool nur mit Hilfe der Warteschlangen des Queue Managers realisiert werden.

3.4.2.2 Senden und Empfangen von Nachrichten

Die zentrale Aufgabe des Lastgenerierungstools ist das Senden von Nachrichten an den Propagationsmanager und das Empfangen der durch den Propagationsmanager abgearbeiteten Nachrichten.
Zur Erfassung der Bearbeitungszeit einzelner Datenänderungspropagationen sollte das Lastgenerierungstool in der Lage sein einzelne Nachrichten an den Propagationsmanager zu senden. Zur Erfassung des dynamischen Verhaltens des Systems sollte das Tool in der Lage sein, mehrere Nachrichten in regelmäßigen Zeitabständen zu senden.

3.4.2.3 Verwaltung von XML-Nachrichten

Die XML-Nachrichten, die an den Propagationsmanager gesendet werden, können entweder durch das Tool zufällig generiert oder vor Beginn der Messexperimente vorgefertigt werden. Die erste Alternative wäre von Vorteil, wenn zur Durchführung der Messungen viele unterschiedliche Nachrichten benötigt werden. Die Verwendung vieler unterschiedlicher Nachrichten wird allerdings die Analyse der Messergebnisse erschweren. Außerdem wird das Reproduzieren der Messexperimente dadurch nicht gewährleistet. Aus diesem Grund wird in dieser Arbeit auf automatische Generierung der Nachrichten durch das Lastgenerierungstool verzichtet. Die Nachrichten werden deswegen in Bezug auf die Messziele der einzelnen Testfälle vorgefertigt und gespeichert. Das Lastgenerierungstool soll dann in der Lage sein die vorgefertigten XML-Nachrichten zu speichern. Weiterhin sollte es bereits gespeicherte XML-Nachrichten, die zur Durchführung der Messungen nicht mehr benötigt werden, entfernen können.

3.4.2.4 Messungen

Bei einer losen Kopplung zwischen dem Propagationsmanager und dem Lastgenerierungstool können nur zwei der in Abschnitt 3.3 definierten Messgrößen mit Hilfe des Lastgenerierungstools gemessen werden. Das ist zum einen die Antwortzeit einzelner ausgelosten Datenänderungspropagationen. Zum anderen kann es den Durchsatz des Propagationsmanager messen.
Das Lastgenerierungstool soll in der Lage sein diese zwei Messgrößen zu erfassen.

3.4.3 Funktionale Anforderungen an die Erweiterungen am Propagationsmanager

Um die Leistung des Propagationsmanagers erfassen zu können, muss seine Funktionalität erweitert werden. Eine Zusammenfassung der Messgrößen wird in Abbildung 3.6 gegeben.

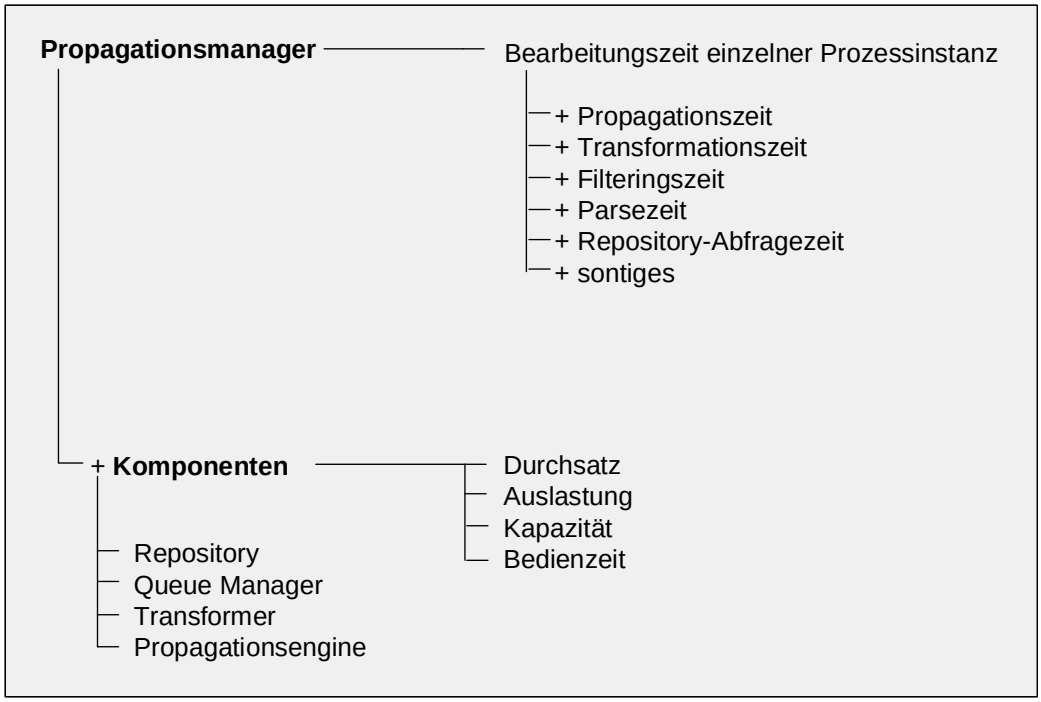

Abbildung 3.6: Zusammenfassung der Messgrößen

3.4.4 Sonstige Anforderungen an das Lastgenerierungstool

Im Folgenden werden die sonstigen, nicht-funktionalen Anforderungen an das Lastgenerierungstool beschrieben.

3.4.4.1 Anforderungen an die Bedienschnittstelle

Eine Bedienschnittstelle ist für das Lastgenerierungstool erforderlich, um das Tool steuern zu können. Da das Lastgenerierungstool hauptsächlich zur Durchführung von Messexperimenten zur Leistungssteigerung des Propagationsmanagers verwendet wird, sind die Benutzer des Lastgenerierungstools hauptsächlich die Entwickler des Propagationsmanagers. Eine graphische Oberfläche zur Steuerung und Administrierung des Tools wurde daher nicht gefordert.
Um das Senden und Verwalten der XML-Nachrichten steuern zu können, sollte eine einfache kommandozeilenorientierte Bedienschnittstelle realisiert werden.

3.4.4.2 Anforderungen an die Programmiersprache

Das Lastgenerierungstool sollte möglicht plattformunabhängig realisiert werden. Bezüglich der Plattformunabhängigkeit sollte sie zumindest auf den Betriebssysteme *Microsoft Windows 2000, XP*, Solaris und Linux lauffähig sein. Um die Plattformunabhängigkeit zu gewährleisten, wird das Lastgenerierungstool in der Programmiersprache *Java* implementiert.

Kapitel 4

Entwurf und Implementierung der Testumgebung

Dieses Kapitel beschäftigt sich mit der Realisierung der Testumgebung. Zunächst wird der Entwurf des Lastgenerierungstools und seiner Komponenten beschrieben. Daraufhin werden die für die Leistungserfassung notwendigen Erweiterungen am Propagationssystem erläutert. Als Nächstes wird auf die Implementierung des Lastgenerierungstools eingegangen und die Bedienung des Tools erläutert.

4.1 Entwurf

In diesem Abschnitt wird die Architektur der Testumgebung beschrieben. Die Hauptkomponenten des Lastgenerierungstools werden definiert und ihre Funktionsweise sowie ihr Zusammenwirken erläutert. Daraufhin wird der objektorientierte Feinentwurf der einzelnen Komponenten beschrieben. Die Klassenstruktur sowie die Attribute und Methoden der wichtigsten Klassen werden dann vorgestellt.
Neben dem Entwurf des Lastgenerierungstools werden auch die Änderungen am Propagationsmanager beschrieben, die die Erfassung der Messdaten ermöglichen.

4.1.1 Komponenten des Lastgenerierungstools

In Abbildung 4.1 ist die Architektur der Testumgebung dargestellt. Sie besteht aus dem Propagationssystem, dessen Leistung gemessen wird, und dem Lastgenerierungstool. Das Lastgenerierungstool erzeugt die Arbeitslast des Propagationssystems, indem es XML-Nachrichten an den Propagationsmanager sendet. Beide Systeme, das Propagationssystem und das Lastgenerierungstool, kommunizieren nur mit Hilfe des Queue Managers. Die gesendeten Nachrichten werden in die Eingabe-Queue des Propagationsmanagers abgelegt. Die transformierten Daten werden an die Ausgabe-Queues gesendet und vom Lastgenerierungstool geholt. Um die Leistungserfassung des Propagationsmanagers zu ermöglichen, wird die Funktionalität des Propagationsmanagers erweitert. Während der Bearbeitung der XML-Nachrichten werden Leistungsdaten für die Arbeit des Systems gesammelt und in Log-Files gespeichert.

Das Lastgenerierungstool besteht aus folgenden Komponenten: den *Konnektoren*, dem *Message Dictionary*, dem *Konnektor Manager* und dem *Command Interpreter*.

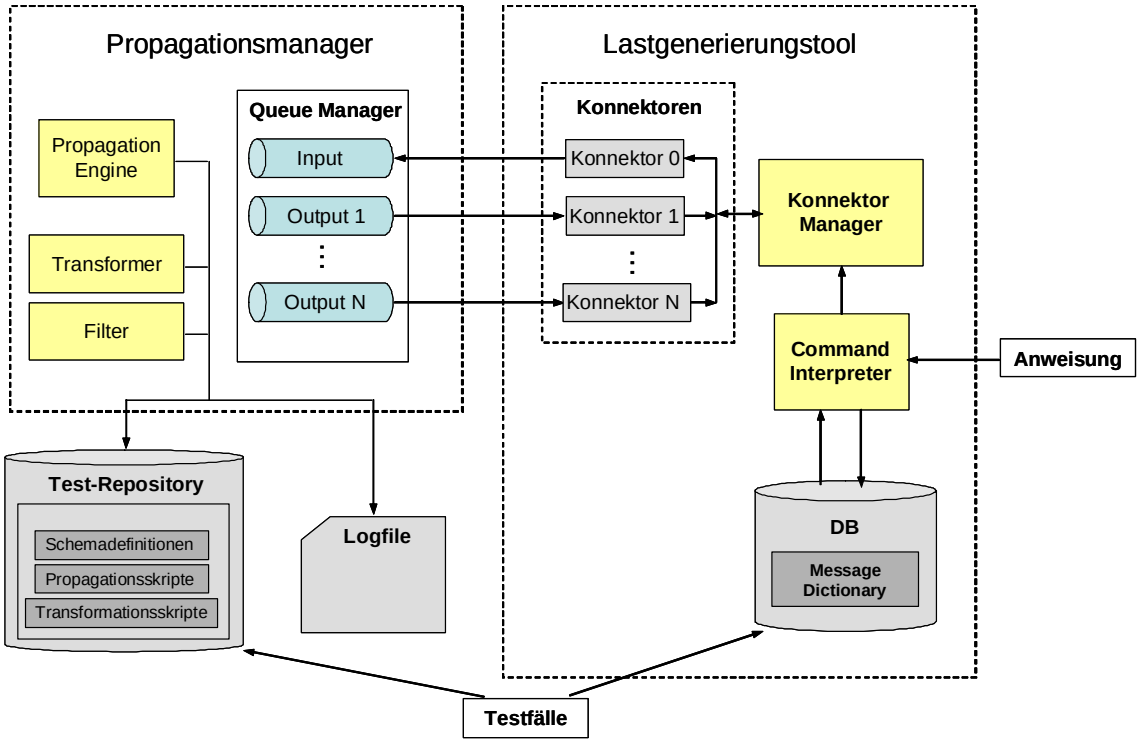

Abbildung 4.1: Leistungsmessung des Propagationssystems

4.1.1.1 Konnektoren

Die *Konnektoren* sind für die Kommunikation zwischen dem Lastgenerierungstool und dem Propagationsmanager verantwortlich. Wenn man diese Architektur mit der Architektur des SIES (siehe Kapitel 2) vergleicht, repräsentieren die Konnektoren die mit dem Propagationssystem verbundenen Informationssysteme. Sie implementieren allerdings nicht die Funktionalität der einzelnen Informationssysteme.
Das Senden einer XML-Nachricht an das Propagationssystem wird durch „Konnektor 0“ durchgeführt. Die anderen Konnektoren sind für das Empfangen der durch den Propagationsmanager propagierten Nachrichten zuständig.

4.1.1.2 Message Dictionary

Die XML-Nachrichten, die durch das Lastgenerierungstool an das Propagationssystem gesendet werden, sind Bestandteil der Testfälle. Eine genaue Definition des Begriffs Testfall wird in Kapitel 5 gegeben. Die Aufgabe des Message Dictionary ist es, diese XML-Nachrichten zu verwalten.
Das Lastgenerierungstool bietet keine Möglichkeit die XML-Nachrichten dynamisch zu generieren, sondern sie müssen bei der Erstellung der Testfälle definiert und in das Message Dictionary gespeichert werden. Der andere Teil der Testfälle, der aus den Schemadefinitionen der Informationssysteme und den Abhängigkeitsdefinitionen, den

XRL+- und XSLT-Skripten, besteht, befindet sich im Repository des Propagationsmanagers.

Das Lastgenerierungstool wird zur Durchführung unterschiedlicher Tests am Propagationsmanager verwendet. Die Anzahl der im Message Dictionary gespeicherten XML-Nachrichten kann deutlich steigen. Um die Übersicht bei der Benutzung der gespeicherten Nachrichten zu verbessern, sollten die XML-Nachrichten in unterschiedliche Gruppen getrennt werden.
Ein weiterer Grund für die Trennung der gespeicherten XML-Nachrichten in unterschiedliche Gruppen ist die Erfassung der dynamischen Charakteristik des Propagationsmanagers. Hierbei können die Nachrichten, die einem Experiment zur Erfassung des dynamischen Verhaltens des Propagationsmanagers gehören, in einer Gruppe zusammengefasst werden.

Um diese Anforderungen zu erfüllen, wird zur Verwaltung der XML-Nachrichten eine Datenbank verwendet.

4.1.1.3 Command Interpreter

Die Arbeit des Lastgenerierungstools wird durch die Eingaben des Benutzers gesteuert. Diese müssen jedoch zuerst interpretiert werden. Diese Aufgabe wird durch den Command Interpreter realisiert. Falls ein Befehl korrekt eingegeben wurde, wird er interpretiert und die entsprechenden Aktionen werden gestartet. Die Befehle können in zwei Gruppen aufgeteilt werden. Die erste Befehlsgruppe bezieht sich auf die Verwaltung von XML-Nachrichten, wie das Hinzufügen von XML-Nachrichten in das Message Dictionary. Falls ein solcher Befehl eingegeben wird, erfolgt die Ausführung durch den Command Interpreter. Die zweite Befehlsgruppe unterstützt das Senden von Nachrichten an den Propagationsmanager. Falls ein solcher Befehl eingegeben wird, werden die benötigten XML-Nachrichten durch den Command Interpreter aus dem Repository geholt und an den Konnektor Manager übergeben.

4.1.1.4 Konnektor Manager

Die Aufgabe des Konnektor Managers besteht hauptsächlich darin, die Antwortzeiten einzelner Datenänderungspropagationen zu ermitteln. Um dieses Ziel zu erreichen, wird die Arbeit der Konnektoren durch den Konnektor Manager koordiniert.
Falls eine Nachricht an den Propagationsmanager gesendet werden soll, wird sie durch den Konnektor Manager an den „Konnektor 0“ übergeben. Empfängt ein Konnektor eine Nachricht, wird sie an den Konnektor Manager übergeben. Der Konnektor Manager speichert Informationen über die gesendeten Nachrichten, so dass er eine empfangene Nachricht einer bereits gesendeten Nachricht zuordnen kann.
Damit letzteres erreicht werden kann, müssen die Nachrichten mit eindeutigen Bezeichnern „markiert“ werden. Beim Empfangen der transformierten Daten wird der Bezeichner der empfangenen Nachrichten mit den Bezeichnern der bereits gesendeten Nachrichten verglichen.
Markiert werden tatsächlich nicht die XML-Nachrichten, sondern die durch die Konnektoren gesendeten JMS-Nachrichten. Für diese Zwecke wird ein weiteres Property-Feld verwendet, um den eindeutigen Bezeichner zu speichern. Damit die durch

den Propagationsmanager propagierten Nachrichten dasselbe Property-Feld wie die empfangene Nachricht haben, muss auch der Propagationsmanager angepasst werden.

Damit eine komplette Änderungspropagation durch den Konnektor Manager erkannt werden kann, muss noch die Anzahl der propagierten Nachrichten bekannt sein. Das kann realisiert werden, indem für jede im Message Dictionary vorhandene XML-Nachricht die Anzahl der propagierten Nachrichten gespeichert wird.

4.1.2 Entwurf der Komponenten

Im Folgenden wird der Entwurf der einzelnen Komponenten des Lastgenerierungstools beschrieben.

4.1.2.1 Konnektoren

Die Aufgabe der Konnektoren bestehen darin Nachrichten an das Propagationssystem zu senden und die durch das Propagationssystem transformierten Datenänderungen zu empfangen. Die Konnektoren werden durch die Klasse *Connector* realisiert. Bei der Initialisierung einer *Connector*-Instanz wird festgelegt, ob diese Instanz Nachrichten an die Eingabe-Queue des Propagationsmanagers senden oder Nachrichten von einer anderen Queue empfangen wird. Im ersten Fall wird während der Initialisierung der Name der Eingabe-Queue übergeben, ansonsten wird der Name der Ausgabe-Queue übergeben, von der die *Connector*-Instanz Nachrichten empfangen wird.
Die Namen der Queues werden mit Hilfe einer Property-Datei administriert (siehe Abschnitt 4.2.4.3 Konfiguration der Konnektoren).

4.1.2.2 Konnektor Manager

Der Konnektor Manager wird durch die Klasse *ConnectorManager* umgesetzt. Eine vereinfache Darstellung der Klasse ist in Abbildung 4.2 dargestellt.

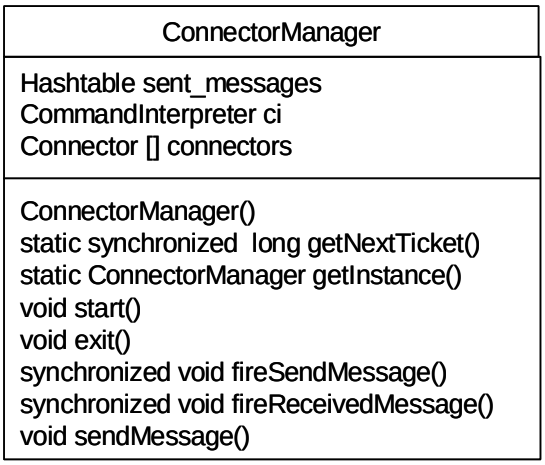

Abbildung 4.2: Klasse ConnectorManager (vereinfacht)

Eine Instanz der Klasse wird durch den Konstruktor der Klasse initialisiert. Die Property-Datei für die Initialisierung der Konnektoren wird ausgelesen und die Konnektoren werden durch den Aufruf der Methode *start()* gestartet. Weiterhin wird eine Instanz der Klasse *CommandInterpreter* (*ci*) initialisiert, damit die Befehle der Benutzer interpretiert werden können. Durch Aufruf der Methode *exit()* wird die Arbeit des „Konnektor Manager“ beendet. Diese Methode wird durch die Instanz des Command Interpreters aufgerufen, wenn der Benutzer einen *exit*-Befehl eingegeben hat. Eine neue Nachricht an den Propagationsmanager wird durch die Methode *sendMessage()* gesendet. Der Konnektor, der für das Senden verantwortlich ist, ruft die Methode *fireSendMessage()*, sobald er die Nachricht gesendet hat. Der Zeitpunkt des Sendens und der eindeutige Bezeichner dieser Nachricht werden in die Hash-Tabelle *sent_messages* gespeichert. Wenn ein Konnektor eine Nachricht empfängt wird die Methode *fireReceivedMessage()* aufgerufen, um den Konnektor Manager zu benachrichtigen.

4.1.2.3 Command Interpreter

Die Komponente Command Interpreter des Lastgenerierungstools wird durch die Klasse *CommandInterpreter* (siehe Abbildung 4.3) realisiert. Nach dem Start des Konnektor Managers wird durch den Konstruktor *CommandInterpreter(ConnectorManager cm)* eine Instanz dieser Klasse initialisiert. Es wird die Datenbankverbindung mit dem Message Dictionary aufgebaut und der Command Interpreter wartet auf Benutzerbefehle. Bei Eingabe eines Befehls wird die Methode *interpret()* aufgerufen, um die eingegebenen Parameter zu parsen. Als Nächstes wird die *execute()* Methode der entsprechenden *Command*-Klasse (siehe Tabelle 4.1) ausgeführt, um die daraus resultierende Aktion zu starten.

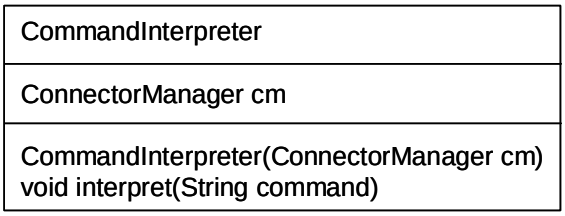

Abbildung 4.3: Klasse CommandInterpreter (vereinfacht)

Klasse	Aufgabe
AddGroupCommand	Eine neue Gruppe ins Message Dictionary hinzufügen
AddMessageCommand	Eine neue Nachricht ins Message Dictionary hinzufügen
AddMessageToGroupCommand	Eine Nachricht in eine Gruppe hinzufügen
DeleteGroupCommand	Eine existierende Gruppe entfernen
DeleteMessageCommand	Eine im Message Dictionary existierende Nachricht entfernen
ExitCommand	Das Lastgenerierungstool beenden
HelpCommand	Hilfe ausgeben
ListGroupCommand	Eine Liste aller Gruppen ausgeben
ListMessageCommand	Eine Liste aller Nachrichten ausgeben
ShowCommand	Den Inhalt einer im Message Dictionary gespeicherten Nachricht ausgeben
ShowGroupCommand	Aller Nachrichten, die einer Gruppe angehören, ausgeben
SimulateCommand	Nachrichten an den Propagationsmanager für die Erfassung der dynamischen Charakteristik senden
StartCommand	Eine einzelne Nachricht an den Propagationsmanager senden

Tabelle 4.1: Klassen zur Umsetzung der Command Interpreter

4.1.2.4 Message Dictionary

Das Message Dictionary ist für die Verwaltung der für die Leistungserfassung verwendeten XML-Nachrichten zuständig. Wie bereits beschrieben, wird für diese Zwecke eine Datenbank verwendet. Im Folgenden wird der konzeptionelle und logische Entwurf des Datenbankschemas diskutiert und anschließend auf die Klassen zur Umsetzung des Message Dictionary eingegangen.

Konzeptioneller und logischer Entwurf des Datenbankschemas

In Abbildung 4.4a) ist der Entwurf des konzeptionellen Datenbankschemas in einem Entity-Relationship Diagramm dargestellt.
Eine *Message* in diesem Schema ist eine XML-Nachricht. Eine XML-Nachricht kann allerdings von mehreren Quellsystemen verwendet werden. Eine SystemMessage ist in diesem Schema eine Nachricht, die durch ein bestimmtes Quellsystem verwendet wird. Mehrere SystemMessages können einer Gruppe angehören und umgekehrt kann eine SystemMessage in mehreren Gruppen enthalten sein.

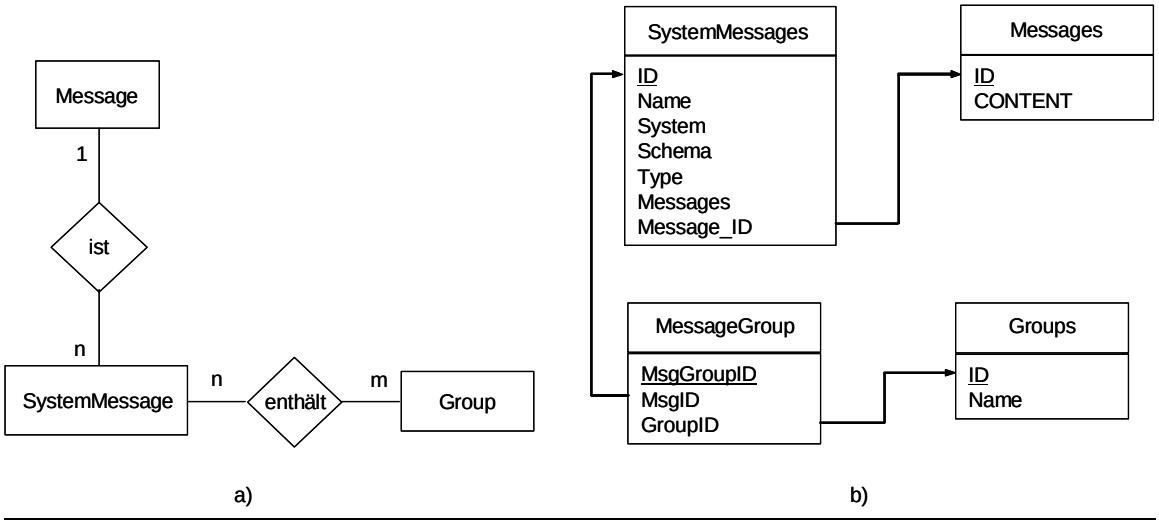

Abbildung 4.4: Konzeptionelles und logisches Datenbankschema des MessageDictionary

In Abbildung 4.4 b) ist dargestellt, wie das konzeptionelle Schema auf das logische Datenbankschema abgebildet wurde. Jede Relation enthält ein Schlüsselattribut, das in der Abbildung unterstrichen ist. Die Fremdschlüsselbeziehungen zwischen den Relationen sind durch Pfeile dargestellt. Die Entities *Message*, *SystemMessage* und *Group* des konzeptionellen Schemas werden direkt auf die Relationen *SystemMessages*, *Messages* und *Groups* abgebildet. In der Relation *MessageGroup* sind für jede Gruppe alle darin enthaltenen SystemMessages aufgeführt, d.h. die Tupel dieser Relation repräsentieren alle möglichen SystemMessages-Group Paare.

Umsetzung, Klassen

Die Verbindung zur Datenbank sowie die Verwaltung der Datenbankobjekte werden durch die Klasse *MessageDictionary* umgesetzt. Die statische Methode *getMessageDictionary()* liefert eine Instanz dieser Klasse. Nach dem ersten Aufruf dieser Methode wird die Verbindung zur Datenbank aufgebaut.
Die Verbindung zur Datenbank kann durch den Aufruf der Methode *closeConnection()* getrennt werden. Die Methoden *getAllSystemMessageNames()* und *getAllGroups()* liefern die Namen aller in der Datenbank gespeicherten Nachrichten bzw. Gruppen. Weiterhin kann man mit Hilfe der Methode *addSystemMessage()* eine neue SystemMessage in das MessageDictionary einfügen. Durch Aufruf der Methode *addSystemMessageToGroup()* wird eine im MessageDictionary vorhandene SystemMessage einer existierenden Gruppe zugeordnet. Eine gespeicherte *SystemMessage* kann mit *getSystemMessage()* und eine Gruppe mit *getGroup()* abgefraget werden.

MessageDictionary
synchronized MessageDictionary getMessageDictionary() void closeConnection() String[] getAllSystemMessageNames() String[] getAllGroups() SystemMessage getSystemMessage(String message) Group getGroup(String group) void addGroup(String groupName) void addSystemMessage(SystemMessage message) void addSystemMessageToGroup(String msgsysName, String groupName) void deleteSystemMessage(String msgsysName) void deleteGroup(String groupName) void deleteAllSystemMessages() void deleteAllMessages() void deleteAllGroups()

Abbildung 4.5: Die Klasse MessageDictionary (vereinfacht)

Eine gespeicherte System-Nachricht kann mit Hilfe der Methode *deleteSystemMessage()* aus der Datenbank entfernt werden, eine existierende Gruppe dementsprechend durch die Methode *deleteGroup()*. Durch den Aufruf der Methoden *deleteAllSystemMessages(), deleteAllMessages()* und *deleteAllGroups()* kann man eine SystemMessage, alle Nachrichten und alle gespeicherte Gruppen aus der Datenbank entfernen.

4.1.3 Erweiterungen am Propagationsmanager

Im Folgenden werden die Erweiterungen am Propagationsmanager beschrieben, die für die Durchführung der Messungen notwendig sind.

4.1.3.1 Erkennung von bearbeiteten Nachrichten

Wenn eine Nachricht durch das Lastgenerierungstool an den Propagationsmanager gesendet wird, wird dem Property-Feld *msg_id* dieser Nachricht ein eindeutiger Schlüsselwert zugewiesen. Damit das Lastgenerierungstool alle propagierten Nachrichten einer gesendeten Nachricht erkennen kann, muss gewährleistet sein, dass das Property-Feld der propagierten Nachrichten den gleichen Property-Wert der gesendeten Nachricht hat.
Diese Anforderung kann durch Änderung des Propagationsmanagers erfolgen. Beim Eintreffen einer Nachricht in die Eingabe-Queue des Queue Managers wird der Wert des Property-Felds *msg_id* ausgelesen. Der Wert dieses Property-Felds wird beim Propagieren der transformierten Daten in das Property-Feld der neuen Nachrichten geschrieben.

4.1.3.2 Log-Files

Die gemessenen Leistungsdaten des Propagationsmanagers werden in Log-Files gespeichert. Auf diese Weise können sie nach der Beendigung der Messexperimente analysiert werden.
Zwei Log-Files werden während der Durchführung eines Messexperiments geführt. Das erste erfasst die Bestandteile der Bearbeitungszeit einzelner Prozessinstanzen, wie in Kapitel 3 beschrieben wurde. Die Struktur dieses Log-Files ist in Abbildung 4.6 dargestellt. Jeder Eintrag dieses Log-Files bezieht sich auf die Bearbeitungszeit einer Prozessinstanz. Er besteht aus einem Bezeichner, der eine Abkürzung der aufgerufenen Methode ist, dem Startzeitpunkt dieser Methode und ihrer Ausführungsdauer. Der Startzeitpunkt bezieht sich auf den Initialisierungszeitpunkt der jeweiligen Prozessinstanz. Die Prozessinstanz wird durch die *prozess_id* identifiziert.

```
getSchemaId 260ms
getXrlText 170ms
getSchemaId 40ms
parsen 152ms
create connector 20ms
init xslt-processor 140ms
nach process init 160ms

<process id="Schema_1_1_1K(1)_1081089419411">
Dependencies start=0 time=170ms
connector start start=1502 time=0ms
open_queue start=1532 time=10ms
message send start=1542 time=120ms
propagate data_in start=1532 time=150ms
process execution start=1502 time=180ms
connector end start=1682 time=0ms
```

Abbildung 4.6: Struktur des ersten Log-Files

Das zweite Log-File ist für die Speicherung der Daten für die dynamische Charakteristik des Propagationsmanagers verantwortlich. Das Log-File besteht aus zwei Teilen (siehe Abbildung 4.7). Der erste Teil zeichnet die wichtigsten, während des Messexperiments aufgerufenen Methoden auf. Es werden der Zeitpunkt des Aufrufs, gemessen vom Beginn des Experiments, die Komponente des Propagationsmanagers, deren Dienste beansprucht werden, die Anzahl der laufenden Operationen auf dieser Komponente und eine Bemerkung, die die Parameter der aufgerufenen Methode enthält, gespeichert.
Der zweite Teil dieses Log-Files enthält Informationen über die Auslastung, die gesamte Bearbeitungszeit und die Kapazität einzelner Komponenten. Zusätzlich wird die Anzahl der durch die einzelnen Komponenten abgearbeiteten Aufträge gespeichert.

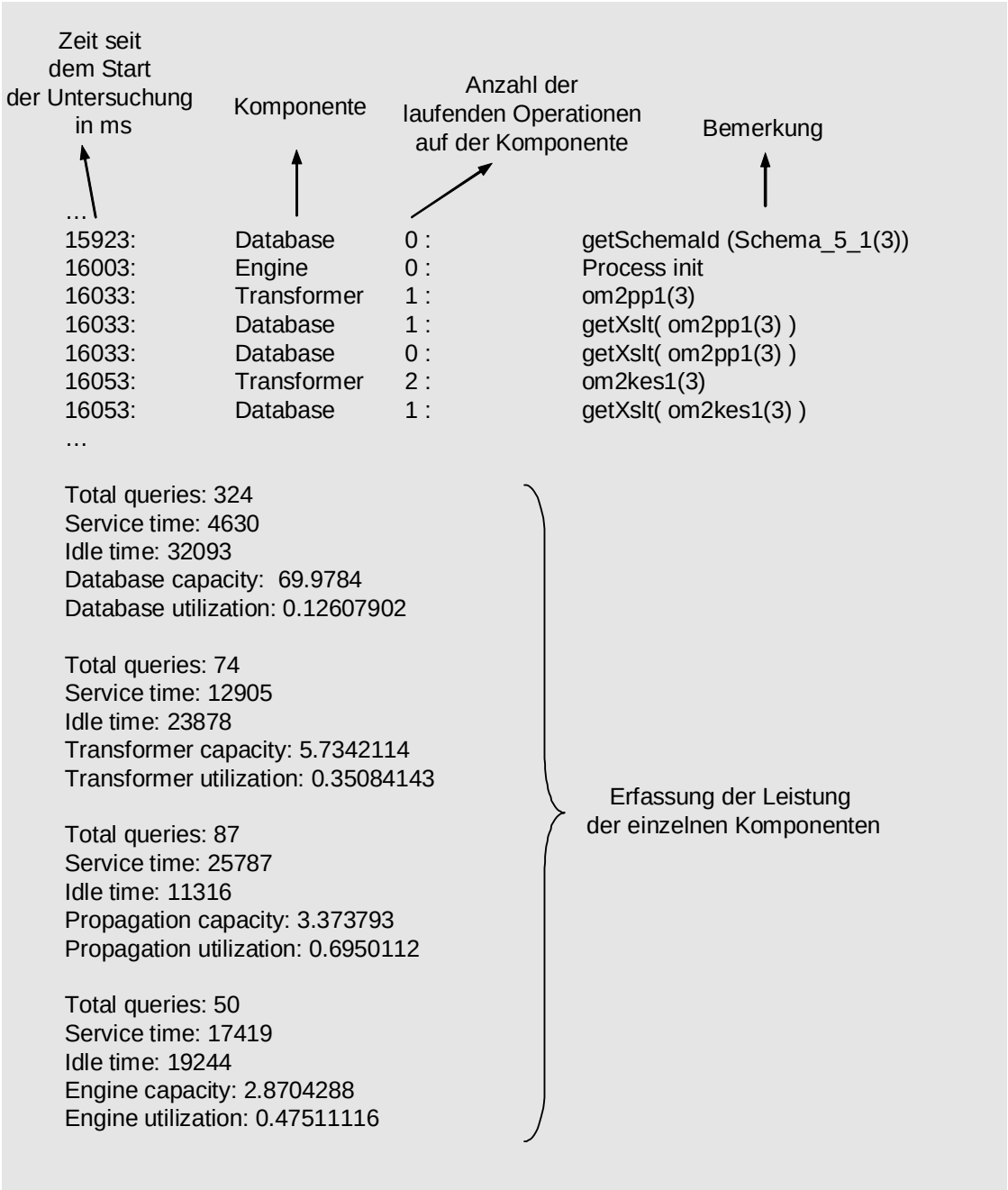

Abbildung 4.7: Struktur des zweiten Log-Files

4.2 Implementierung

Im Folgenden wird die Implementierung der Testumgebung des Propagationsmanagers diskutiert. Zunächst wird auf die Implementierungsumgebung eingegangen. Anschließend wird die Datenbank vorgestellt und die Installation und Benutzung der Testumgebung werden erklärt.

4.2.1 Rahmenbedingungen

Die Entwicklung der Testumgebung erfolgt hauptsächlich auf einem Rechner mit dem Betriebsystem *Windows XP Professional*. Als Programmiersprache wird die *Java 2 Plattform, Standard Edition Version 1.4.1* (J2SE 1.4.1) verwendet.

Aufgabengebiet	System/Werkzeug
Betriebsystem	Windows XP Professional
Implementierung	Java 2 Standard Edition 1.4.1, Java 2 Enterprise Edition 1.3.1 inc. Referenzimplementierung des Java Message Service 1.3
Entwicklungsumgebung	Java Forte 3.0
Datenhaltung	DB2 Version 8.1
Hardware	PC mit CPU Intel Pentium 4 2GHz, 256MB PC mit CPU Intel Pentium 3 600MHz, 320 MB

Tabelle 4.2: Implementierungsumgebung

4.2.2 Die Datenbank

Die für die Durchführung der Testfälle benötigten Nachrichten werden in einer Datenbank verwaltet. Für diese Zwecke wurde das relationale Datenbanksystem *DB2 Version 8.1* von *IBM* verwendet. Die Kommunikation mit der Datenbank wurde mit Standard-SQL realisiert. Da die Konfigurationsdaten zur Aufbau der Verbindung mit der Datenbank mit Hilfe einer Konfigurationsdatei (siehe Abschnitt 4.2.4) umgesetzt wurden, kann dieses Datenbanksystem jederzeit durch ein anderes ersetzt werden. Die eigentliche Kommunikation mit der Datenbank erfolgt durch den JDBC-Triber von *DB2*, der mit der Datei *db2java.zip*, die sich im Installationsverzeichnis des Datenbanksystems befindet, mitgeliefert wird. Für das Ausführen der Testumgebung wird dieser Treiber benötigt.

4.2.3 Dokumentation

Die Struktur des Quellcodes des Lastgenerierungstools sowie die Änderungen am Propagationsmanager wurden mit zahlreichen Kommentaren verdeutlicht. Im Quellcode sind alle Klassen und Public-Methoden so kommentiert, dass mit dem Werkzeug *Javadoc* eine umfassende und standardisierte HTML-Dokumentation erzeugt werden kann. Zu jeder Methode werden die Parameter, der Rückgabewert und die Exceptions, die auftreten können, aufgelistet und kommentiert. Die resultierende Dokumentation verdeutlicht die Struktur des Quellcodes und bietet einen Vorteil bei der Weiterentwicklung des Systems.

4.2.4 Betrieb

4.2.4.1 Installation und Betrieb des Java Message Services

Für den Betrieb der Software sind die Java Message Services notwendig, die sich in dem Java 2 Enterprise Edition SDK befinden. Dafür muss auch der Pfad auf das Installationsverzeichnis korrekt gesetzt werden. Der Start erfolgt über den Befehl:

```
j2ee -verbose
```

Vor dem Start des Propagationsmanagers und der Testumgebung müssen die Nachrichten-Queues eingerichtet werden. Falls das noch nicht geschehen ist, sind folgende Schritte durchzuführen:

- Durch den folgenden Befehl wird ein ConnectionFactory-Objekt eingerichtet, das zum Aufbau einer Verbindung mit der Eingabe-Queue benötigt wird:

```
J2eeadmin –addJmsFactory PMInputConnectionFactory topic –props
client_id=PMClient
```

- Die Eingabe-Queue für den Propagationsmanager wird eingerichtet durch den Befehl:

```
j2eeadmin –addJmsDestination PMInput topic
```

- Die Ausgangs-Queues werden eingerichtet mit den Befehlen:

```
j2eeadmin –addJmsDestination QUEUE_SYSTEM1 queue
j2eeadmin –addJmsDestination QUEUE_SYSTEM2 queue
j2eeadmin –addJmsDestination QUEUE_SYSTEM3 queue
…
```

Die Namen *QUEUE_SYSTEM1*, *QUEUE_SYSTEM2*,… müssen mit den Namen der Testsysteme, die im Repository hinterlegt sind, übereinstimmen.

Bevor das Lastgenerierungstool in Betrieb genommen wird, müssen die Datenbank und die Konnektoren der Testumgebung konfiguriert werden. Die gesamte Konfiguration erfolgt mit Hilfe der Konfigurationsdatei *testtool.properties*.

4.2.4.2 Konfiguration der Datenbank

Die Konfiguration der für die Speicherung der Testnachrichten verwendeten Datenbank erfolgt durch folgende Einträge der Konfigurationsdatei des Lastgenerierungstools.

```
dbUrl  =  jdbc:db2:sies
dbUser  =  <username>
dbPass  =  <password>
dbDriver  =  COM.ibm.db2.jdbc.app.DB2Driver
```

Der Eintrag *dbUrl* definiert den Namen der Datenbank, *dbUser* und *dbPass* werden von der Anwendung benötigt, um eine Verbindung mit dem Datenbanksystem aufzubauen.

Der Eintrag *dbDriver* spezifiziert den Namen des JDBC-Treibers. Durch Änderung von *dbUrl* und *dbDriver* kann das verwendete Datenbanksystem durch ein anderes ersetzt werden.

4.2.4.3 Konfiguration der Konnektoren

Damit das Lastgenerierungstool Nachrichten an das Propagationssystem senden und Nachrichten vom Propagationssystem empfangen kann, müssen die Konnektoren des Lastgenerierungstools konfiguriert werden. Die Konfiguration der Konnektoren wird anhand des folgenden Beispiels verdeutlicht.

```
CONNECTOR_NUMBER = 6

CONNECTOR_0 = PMInput
CONNECTOR_1 = QUEUE_SYSTEM_1
CONNECTOR_2 = QUEUE_SYSTEM_2
CONNECTOR_3 = QUEUE_SYSTEM_3
CONNECTOR_4 = QUEUE_SYSTEM_4
CONNECTOR_5 = QUEUE_SYSTEM_5
```

Die Anzahl der zu initialisierenden Konnektoren wird durch den Parameter *CONNECTOR_NUMBER* spezifiziert.
Der erste Konnektor wird verwendet, um Nachrichten an das Propagationssystem zu senden. Durch den Eintrag *CONNECTOR_0* wird der Namen des Topics definiert, das zum Senden der Nachrichten verwendet wird (standardmäßig ist PMInput der Name des Topics. Aus diesem Grund wird *CONNECTOR_0* auf PMInput gesetzt). Die anderen Einträge, die alle im Format *CONNECTOR_$N* sind, wobei *$N* eine natürliche Zahl ist, definieren die Namen der Queues, die von den anderen Konnektoren benutzt werden, um die transformierten Nachrichten zu empfangen. Die Namen der verwendeten Queues müssen den im Repository hinterlegten Namen der Queues entsprechen. Falls die konfigurierten Nachrichten-Queues nicht existieren, wird die Arbeit des Lastgenerierungstools beendet.

4.2.4.4 Starten des Propagationsmanagers

Bevor die Testumgebung gestartet wird, muss der Propagationsmanager durch den folgenden Befehl gestartet werden [KO01]:

```
java pm.PropagationManager
–D jms.properties=<j2ee-pfad>/config/jms.properties <host name> pm
1099
```

4.2.4.5 Starten des Lastgenerierungstools

Um das Lastgenerierungstool zu starten, wird die Hauptfunktion der Klasse *ConnectorManager* ausgeführt. Es ist dabei lediglich die Eingabe der folgenden Zeile im Root-Verzeichnis der Testumgebung erforderlich:

```
java -cp .; db2java.zip de/uni_stuttgart/sfb467/let/ConnectorManager
```

Durch den Parameter ‘cp’ wird der *Classpath* angegeben, dieser repräsentiert den Suchpfad für zusätzliche Dateien. Die ZIP-Datei *db2java.zip* enthält die JDBC-Treiber, die von der Anwendung benötigt wird, um die Verbindung mit der Datenbank aufzubauen.

4.2.4.6 Bedienung des Lastgenerierungstools

Nachdem das Lastgenerierungstool gestartet ist, kann der Benutzer die Arbeit des Tools durch Eingabe von Befehlen steuern. Im Folgenden werden die Befehle und Parameter der Bedienschnittstelle aufgelistet, mit deren Hilfe das Lastgenerierungstool gesteuert wird.

- Durch den folgenden Befehl kann eine neue Gruppe erstellt werden:

  ```
  addGroup <group name>
  ```

  Durch den Parameter *group name* wird der Name der Gruppe übergeben (maximal 100 Zeichen).

  Eine Fehlermeldung wird zurückgegeben, wenn die Gruppe schon existiert oder wenn der Name mehr als 100 Zeichen (eine Beschränkung ist erforderlich, da die Daten in einer Datenbank gespeichert werden) enthält.

- Eine neue Systemnachricht wird hinzugefügt mit dem Befehl:

  ```
  addMessage -f <file name> -s <schema name> -sys <system name>
             -n <message name> -m <number of messages>
  ```

  file name: Name der Datei, die die XML-Nachricht enthält.
  schema name: Name des Quellschemas (maximal 100 Zeichen).
  system name: Name des Quellsystems (maximal 100 Zeichen).
  message name: Name, unter der diese Nachricht in das MessageDictionary gespeichert werden soll (maximal 100 Zeichen).
  number of messages: Anzahl der Nachrichten, die an die Zielsysteme weitergeleitet werden.

- Eine Nachricht kann in eine Gruppe hinzugefügt werden mit dem Befehl:

  ```
  addMessageToGroup –m <message name> -g <group name>
  ```

  message name: Name der Nachricht.
  group name: Name der Gruppe.

  Falls der Name der Nachricht oder der Name der Gruppe nicht existiert, wird eine Fehlermeldung ausgegeben.

- Eine bereits vorhandene Gruppe wird durch folgenden Befehl gelöscht:

  ```
  deleteGroup <group name>
  ```

  group name: Name der Gruppe

Falls die Gruppe nicht existiert, wird eine Fehlermeldung ausgegeben.

- Eine bereits vorhandene Nachricht wird durch folgenden Befehl gelöscht:

  `deleteMessage <message name>`

  message name: Name der Nachricht.

  Falls die Nachricht nicht existiert, wird eine Fehlermeldung ausgegeben.

- Der Benutzer kann sich jederzeit eine Hilfeseite vom Tool anzeigen lassen durch Eingabe des Befehls:

  `help [<command name>]`

  command name: Name des Befehls. Falls kein Name eingegeben wird, werden Informationen über alle unterstützten Befehle ausgegeben.

- Alle gespeicherten Nachrichten werden aufgelistet mit dem Befehl:

  `listMessages`

- Die Namen aller gespeicherten Gruppen werden aufgelistet mit dem Befehl:

  `listGroups`

- Der Inhalt einer gespeicherten XML-Nachricht wird angezeigt durch folgenden Befehl:

  `show <message name>`

  message name: Name der Nachricht, unter dem sie im MessageDictionary gespeichert ist.

- Die Namen aller Nachrichten, die einer Gruppe angehören werden aufgelistet mit dem Befehl:

  `showGroup <group name>`

  group name: Name der Gruppe.

- Der folgende Befehl sendet einige Nachrichten nacheinander.

  `simulate -num <number of messages> -time <time>`

  number of messages: Anzahl der an den Propagationsmanager zu sendenden Nachrichten.
  time: Zeitintervall zwischen den einzelnen Sendezeitpunkten.

- Eine einzelne Nachricht wird an den Propagationsmanager gesendet mit dem Befehl:

  `start <message name>`

  message name: Name der im Message Dictionary gespeicherten Nachricht.

Kapitel 5

Testfälle und Messergebnisse

In diesem Kapitel werden die Testfälle beschrieben, die zur Untersuchung des Propagationssystems verwendet werden.

5.1 Bestandteile der Testfälle

Die Testfälle in dieser Arbeit werden erstellt, um die Leistungsprobleme des Propagationsmanagers zu finden. Durch die Definition eines Testfalls wird eine bestimmte Arbeitslast für den Propagationsmanager festgelegt. In Kapitel 2 wurde bereits beschrieben, dass die Arbeitslast des Propagationsmanagers folgende Bestandteile hat:

- Eine Menge von XRL+-Dokumenten, die die Abhängigkeiten zwischen Quell- und Zielsystemen definiert.
- Eine Menge von XSLT-Dokumenten, die innerhalb der XRL+-Dokumente referenziert werden.
- Eine Menge von XML-Nachrichten, die die Propagationsprozesse im Propagationsmanager auslösen.
- Zeitliche Charakteristik des Eintreffens der XML-Nachrichten im Propagationssystem.

Durch diese Bestandteile wird auch ein Testfall definiert. Um die statische und die dynamische Charakteristik des Propagationsmanagers zu untersuchen, werden die Testfälle in zwei Gruppen unterteilt:

- Die erste Gruppe von Testfällen, von Testfall 1 bis Testfall 3, wurde erstellt, um die Bearbeitungszeit einzelner Datenänderungspropagationen und die einzelnen Bestandteile der Bearbeitungszeit zu untersuchen. Durch die gewonnenen Erkenntnisse können eventuelle Maßnahmen zur Verbesserung des Propagationsmanagers getroffen werden, um die problematischen Bestandteile der Bearbeitungszeit zu optimieren.
  Bei diesen Testfällen werden einzelne XML-Nachrichten gesendet und gewartet bis sie durch den Propagationsmanager abgearbeitet worden sind. Weiterhin

wird bei diesen Testfällen sichergestellt, dass genau eine Prozessinstanz durch diese eine Nachricht ausgelöst wird und der Propagationsmanager höchstens eine Propagationsnachricht bearbeitet.

- Die zweite Gruppe, von Testfall 4 bis Testfall 6, sind kleine Benchmarktests, die zur Untersuchung der dynamischen Charakteristik verwendet werden. Das Ziel dieser Benchmarks ist es, den Propagationsmanager auf mögliche Engpässe zu untersuchen. Um dieses Ziel zu erreichen, werden die unterschiedlichen Komponenten des Propagationsmanagers, der Transformer, das Repository, die Propagationsengine und der Queue Manager, unterschiedlich stark belastet.

5.2 Testfälle

Im Folgenden werden die einzelnen Testfälle definiert.

5.2.1 Testfall 1

Der zweite Testfall untersucht die Eigenschaften der Messaging Middleware (MOM), um den Zeitaufwand einer Propagate-Operation herauszufinden. Für diese Zwecke wird ein XRL+-Dokument, wie in Abbildung 5.1 dargestellt, erstellt und Messungen mit unterschiedlichen Nachrichtengrößen durchgeführt. Der Pfeil in Abbildung 5.1 stellt den Kontrollfluss im XRL+-Dokument dar.

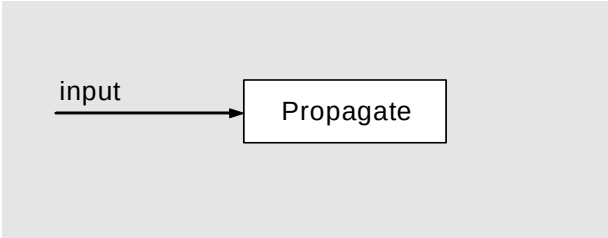

Abbildung 5.1: Struktur des XRL+-Dokuments mit einer Propagate-Operationen

Zur Untersuchung der Eigenschaften der Middleware (im Folgenden wird unter Middleware die Messaging Middleware für den Queue Manager verstanden) werden folgende Nachrichtengrößen vorgeschlagen:

Name	Nachrichtengröße
Testfall 1.1	1 KB
Testfall 1.2	10 KB
Testfall 1.3	100 KB

Tabelle 5.1: Konfigurationsparameter für den Testfall 1

Testfall 1 kann verwendet werden, um die Leistung unterschiedlicher JMS-Server zu vergleichen.

5.2.2 Testfall 2

Um die Leistung des Transformers zu untersuchen und die in Kapitel 6 durchgeführten Optimierungen am Transformer zu bewerten, wurde Testfall 3 erstellt. Die Bestandteile der Bearbeitungszeit werden auch mit Hilfe dieses Testfalls untersucht. Außerdem werden die für diesen Testfall definierten Transformationen (XSLT-Dokumente) auch in den nachfolgenden Testfällen verwendet.

Zur Untersuchung der Leistung des Transformers sollen unterschiedliche Transformationsskripte definiert werden. Eine vollständige Abdeckung der Funktionalität der XSLT-Sprache ist jedoch nicht Ziel dieser Arbeit. Aus diesem Grund wird sich die Untersuchung auf drei unterschiedliche Transformationen beschränken.

Es werden drei XSLT-Dokumente definiert, die, wie in Abbildung 5.3 dargestellt ist, folgende Transformationen (den Quell-Code der XSLT-Dokumente steht in Anhang A) auf die eingegebenen XML-Daten durchgeführt:

- Strukturumwandlung der Daten des XML-Dokuments. Hierbei werden nur die Elementtypen[1] des XML-Dokuments neu benannt. (Transformation 1),
- Sortierung der Werte des XML-Dokuments (Transformation 2)
- und Aggregation der Werte des XML-Dokuments (Transformation 3).

Es könnten auch andere Transformationen zur Bewertung des Transformers definiert werden. Da die drei vorgeschlagenen Transformationen grundlegende Transformationen darstellen, Sortierung, Strukturumwandlung und Aggregation, wurden diese ausgewählt.

Die Struktur der XRL+-Dokumente für diesen Testfall sind in Abbildung 5.2 dargestellt. Um die einzelnen XRL+-Dokumente für die unterschiedliche Transformationen zu unterscheiden, wird das XRL+-Dokument mit der ersten Transformation (Strukturumwandlung) als Testfall 2.1, das zweite als Testfall 2.2 und das dritte als Testfall 2.3 bezeichnet.

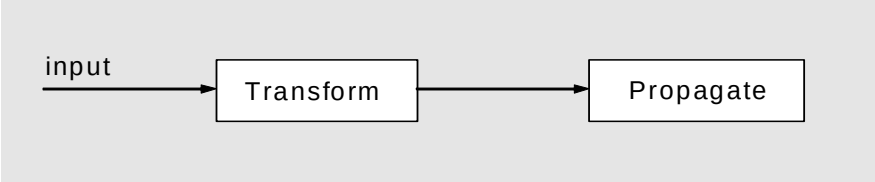

Abbildung 5.2: Struktur der XRL+-Dokumente zur Untersuchung des Transformers

Zur Auswertung der Leistung des Transformers werden drei Nachrichten, wie in Tabelle 5.2 definiert, verwendet. Da alle drei Transformationen so definiert wurden, dass sie auf dieselben XML-Dokumente angewendet werden können, können diese Nachrichten für alle Abhängigkeiten verwendet werden.

[1] Da in den verschiedenen Veröffentlichungen die Begriffe Tag, Element, Elementtyp oft verwechselt werden, wird hier die Definition aus [MS02] verwendet.
Um diese Begriffe zu erklären, betrachte man folgendes Beispiel:
<*name*>*Sara*</*name*>
Dieses Bespiel zeigt ein vollständiges Element, das aus dem Start-Tag (<*name*>), dem Inhalt (*Sara*) sowie End-Tag (</*name*>) besteht. Der Typ des Elements ist *name*. Der Typ bezeichnet die Gattung, die Art des Elements. Ein Element ist ein Vertreter des entsprechenden Typs.

Nachricht	Anzahl Einträge	Dateigröße
Nachricht 1	1	1 KB
Nachricht 2	10	3 KB
Nachricht 3	100	21 KB

Tabelle 5.2: XML-Nachrichten für den Testfall 2

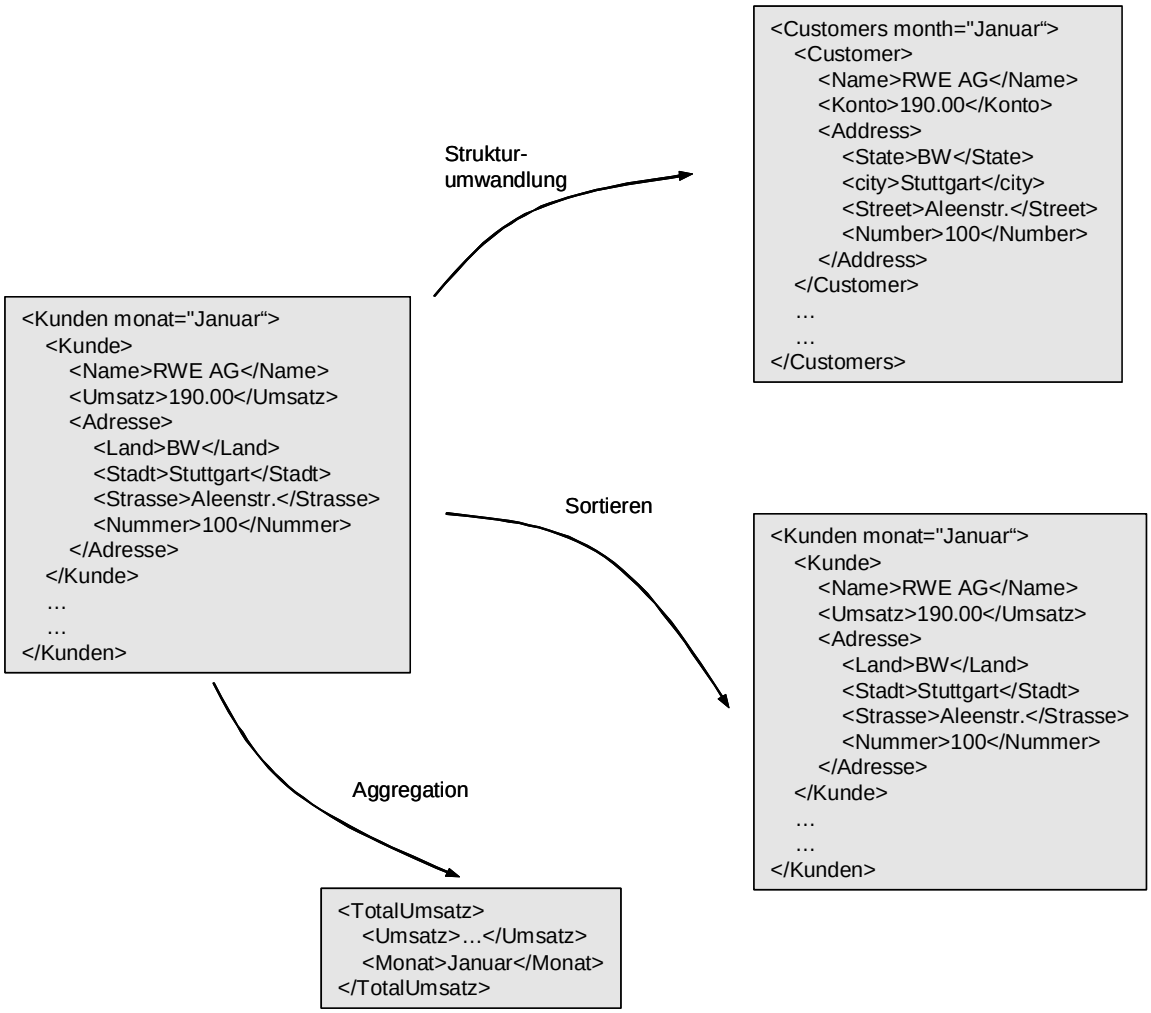

Abbildung 5.3: Beispiele für die Transformationen

5.2.3 Testfall 3

Durch den Testfall 3, der im Folgenden beschrieben wird, kann die Parsezeit der XML+-Dokumente in Abhängigkeit von der Größe des XRL+-Dokuments ermittelt werden. Dabei wird die Größe nicht in Bytes gemessen, sondern durch die Anzahl der Einträge im XRL+-Dokument. Der Inhalt des für diesen Testfall verwendeten XRL+-Dokumente wird in Abbildung 5.4 dargestellt.
Bei diesem Testfall wird die Parsezeit gemessen, die der XML-Parser benötigt, um das entsprechende XRL+-Dokument zu parsen.

```
<?xml version="1.0" encoding="UTF-8"?>
<xrl:ROUTE id="1" xmlns:xrl="http://informatik.uni-stuttgart.de/XRL">
  <xrl:SEQUENCE>
    <xrl:WAIT sync="1">
      <xrl:MESSAGE_EVENT initial="yes" schema="SCHEMA_IN" system="SYSTEM_IN“
       type="any" xml_out=" in_data"/>
    </xrl:WAIT>
    <xrl:TRANSFORM xml_in="in_data" xml_out= "out_data" xslt="xslt_1"/>
    …
    <xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt_1"/>          } N-Mal

    <xrl:PROPAGATE xml="out_data" system="SYSTEM_OUT" schema="SCHEMA_OUT"/>
</xrl:SEQUENCE>
</xrl:ROUTE>
```

Abbildung 5.4: Die Struktur der XRL+-Dokumente

Bei der Erstellung der nächsten Testfälle wurde das Ziel verfolgt, die Arbeit des Propagationsmanagers und seiner Komponenten beim dynamischen Betrieb zu untersuchen. Bei allen nachfolgenden Testfällen werden die unterschiedlichen Komponenten des Propagationsmanagers unterschiedlich stark belastet. Wertvolle Informationen über die Arbeit der einzelnen Komponenten werden durch das Leistungsmaß, Auslastung, geliefert.

Für das Messexperiment für diese Testfälle wird das Zeitintervall zwischen den einzelnen Sendezeitpunkten der Nachrichten auf 100ms festgelegt. Das ist ungefähr die Geschwindigkeit, mit der das Lastgenerierungstool für die verwendete Messkonfiguration (siehe Abschnitt 5.3.1) in der Lage ist.

5.2.4 Testfall 4

Testfall 4 ist der erste Testfall, der die dynamische Charakteristik des Propagationssystems untersucht. Für die Abhängigkeiten und die Transformationen dieses Testfalls wird nichts Neues definiert. Es werden dieselben XRL+- und XSLT-Dokumente aus Testfall 2.1 verwendet und als XML-Nachricht wird Nachricht 1, die in Testfall 2 festgelegt wurde, verwendet.

5.2.5 Testfall 5

Die Idee dieses Testfalls ist es, den Transformer und das Repository zu belasten. Um das zu erreichen, werden mehrere Transformationen innerhalb der Abhängigkeitsdefinitionen definiert.
Die Struktur des XRL+-Dokuments wird in Abbildung 5.5 dargestellt. Durch die Verwendung mehrerer Transformationen innerhalb einer Prozessdefinition werden gleichzeitig der Transformer und das Repository belastet. Durch die Einführung von parallelen Operationen in der Prozessdefinition werden parallele Operationen auf die einzelnen Komponenten des Propagationsmanagers ausgeführt.

Für die Transformation wird Transformation 1 aus Testfall 2 verwendet. Als Propagationsnachricht wird Nachricht 1 verwendet.

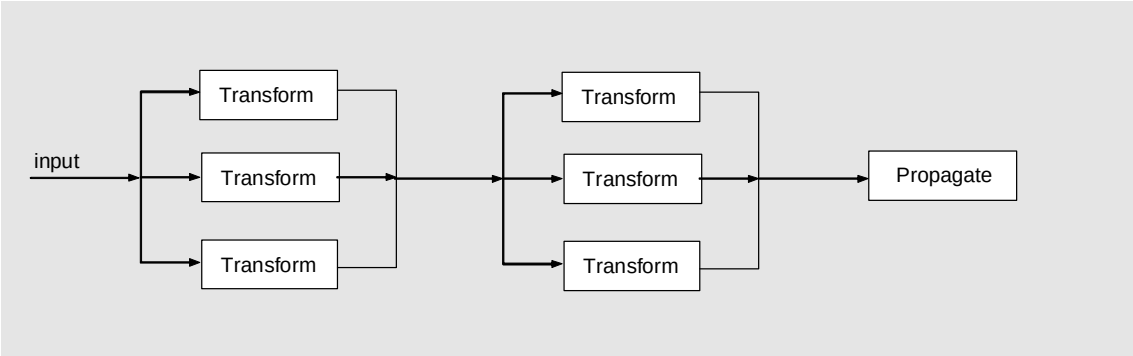

Abbildung 5.5: Struktur der XRL+-Dokumente für den Testfall 5

5.2.6 Testfall 6

Die Struktur des für diesen Testfall definierten Abhängigkeitsskripts ist in Abbildung 5.6. Er besteht aus drei parallelen Transformationen und drei parallelen Propagate-Operationen. Durch diesen Testfall wird das dynamische Verhalten des Propagationsmanagers bei mehreren Propagate-Operationen untersucht. Getestet werden dadurch der Queue Manager, der Transformer und das Repository. Die Struktur des XRL+-Dokuments ist in Abbildung 5.6 dargestellt.
Für die Transformation wird Transformation 1 verwendet und als Propagationsnachricht wird die Nachricht 1 verwendet, die in Testfall 2 festgelegt wurde.

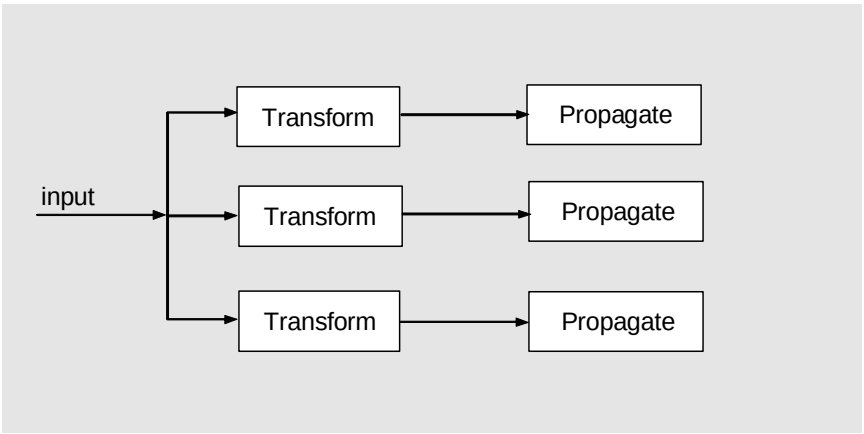

Abbildung 5.6: Struktur des XRL+-Dokuments für den Testfall 6

5.3 Messergebnisse

Die Messergebnisse, die im Folgenden vorgestellt werden, repräsentieren den Stand des Propagationsmanagers zu Beginn dieser Arbeit.

5.3.1 Messkonfiguration

Alle Messungen wurden auf einem Rechner durchgeführt, auf dem das Lastgenerierungstool und der Propagationsmanager mit dem Repository installiert sind. Die Systemkonfiguration ist in Tabelle 5.3 dargestellt.

Betriebsystem	Windows XP Professional
Hardware	PC mit CPU Intel Pentium 4 2GHz, 256MB

Tabelle 5.3: Systemkonfiguration

5.3.2 Testfall 1-3

Die Ergebnisse des Testfalls 1 sind in Abbildung 5.7 dargestellt. Man stellt fest, dass das Messaging-System bei Nachrichtengrößen zwischen 1Kb und 100Kb sehr gut skaliert. Das Propagieren von Nachrichten größer als 100Kb ist allerdings mit sehr hohem Zeitaufwand verbunden. Das Versenden 1MB großer Nachrichten ist sogar teuerer als das Versenden von zehn 100Kb-Nachrichten. Das Propagieren kleiner Nachrichten ist auch eine ungünstige Operation. Sogar die Weiterleitung einer Nachricht mit Größe 1 Byte ist mit Kosten verbunden. Die fixen Kosten, die unabhängig von Nachrichtengröße anfallen, sind in Abbildung 5.7 mit „FIX“ gekennzeichnet.

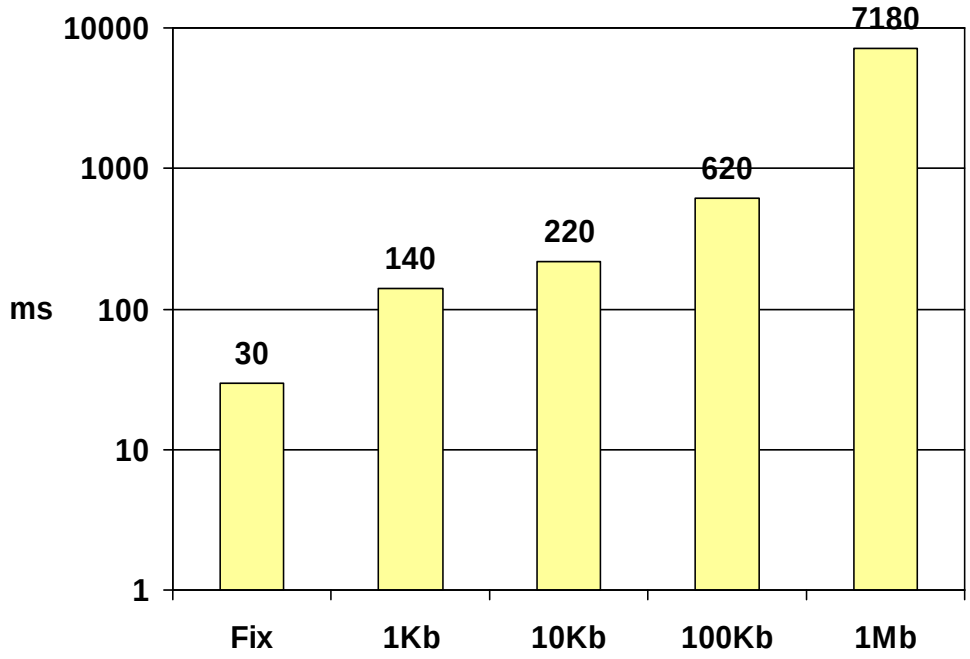

Abbildung 5.7: Kosten für eine Propagation in Abhängigkeit von der Größe der Nachricht

Die Messergebnisse für den Testfall 2 sind in Tabelle 5.4 dargestellt. Die Messdaten zeigen, dass die Initialisierung und die Beendigung der Prozessinstanzen teuere

Operationen sind. Insgesamt benötigt eine Prozessinstanz 260ms zur Initialisierung und noch 120ms, wenn sie beendet wird.

Testfall	Nachricht Anzahl Einträge	Prozessinstanz Initialisierung			Prozessinstanz Ausführen			Prozess-instanz Auflösen
		Rep.-Abfragen	Parsen XRL+	Rest	Transform.	Rep.-Abfragen	Propagate	
Testfall 2.1	1	20ms	20ms	220ms	90ms	10ms	100ms	120ms
Testfall 2.1	10	20ms	20ms	220ms	100ms	10ms	110ms	120ms
Testfall 2.1	100	20ms	20ms	220ms	100ms	10ms	130ms	120ms
Testfall 2.2	1	20ms	20ms	220ms	70ms	10ms	80ms	120ms
Testfall 2.2	10	20ms	20ms	220ms	110ms	10ms	110ms	120ms
Testfall 2.2	100	20ms	20ms	220ms	140ms	10ms	150ms	120ms
Testfall 2.3	1	20ms	20ms	220ms	90ms	10ms	100ms	120ms
Testfall 2.3	10	20ms	20ms	220ms	100ms	10ms	130ms	120ms
Testfall 2.3	100	20ms	20ms	220ms	140ms	10ms	130ms	120ms

Tabelle 5.4: Messergebnisse für den Testfall 2

Die Ergebnisse für den Testfall 3 werden in Abbildung 5.8 dargestellt. Sie zeigen die Abhängigkeit zwischen der Größe eines XRL+-Dokuments und der Parsezeit dieses Dokuments. Die Größe der getesteten XRL+-Dokumente wird durch die Einträge (N) der XRL+-Dokumente angegeben.

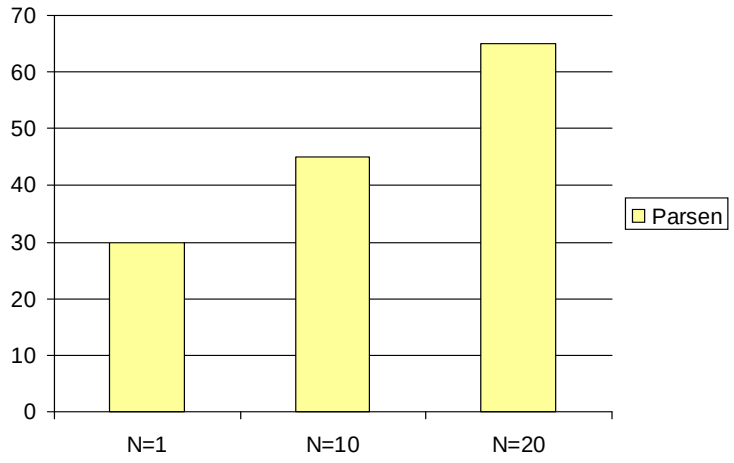

Abbildung 5.8: Parsezeit einer XRL+-Datei

5.3.3 Testfall 4-6

Die Ergebnisse für den Testfall 4 sind in Tabelle 5.5 dargestellt. Sie zeigen deutlich, dass die Propagationsengine mit einer Auslastung von 88% (0.88) die Leistung des Propagationsmanagers bestimmt. Für die weitere Arbeit bedeuten diese Messdaten, dass die Leistung des Propagationsmanagers zunächst durch Optimierung der Propagationsengine gesteigert werden kann. Werden auch die Ergebnisse des Testfalls 3 berücksichtigt, wird ersichtlich, dass die Optimierung der Prozessinitialisierungszeiten und Prozessbeendigungszeiten der erste Schritt zur Leistungssteigerung des Propagationsmanagers sein sollte.

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100	100	902	100
Auslastung	0.88	0.23	0.13	0.27
Kapazität	0.94	4.44	61	3.7
Bearbeitungszeit	84s	23.6s	19.1s	25.3s
Gesamtzeit	95.4s			

Tabelle 5.5: Zusammenfassung der Ergebnisse für den Testfall 4

Die schlechte Leistung der Propagationsengine wird auch durch die Messergebnisse für den Testfall 5 (siehe Tabelle 5.6) und Testfall 6 (siehe Tabelle 5.7) bestätigt. Weiterhin zeigen die Messdaten für den Testfall 5, dass die Repository-Abfragen und die Transformationen einen hohen Zeitaufwand verursachen können.

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100	600	2550	100
Auslastung	0.86	0.48	0.44	0.15
Kapazität	0.70	7.5	34.7	4.00
Bearbeitungszeit	141s	79.1s	73s	25s
Gesamtzeit	164.2s			

Tabelle 5.6: Zusammenfassung der Ergebnisse für den Testfall 5

Die Messdaten für den Testfall 6 zeigen, dass auch die Ausführung vieler Propagate-Operationen einen deutlichen Zeitaufwand verursachen kann.

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100	300	1674	300
Auslastung	0.81	0.42	0.48	0.57
Kapazität	0.91	5.23	26	3.94
Bearbeitungszeit	108.9s	57.3s	63.7s	76s
Gesamtzeit	134.5s			

Tabelle 5.7: Zusammenfassung der Ergebnisse für den Testfall 6

Kapitel 6

Leistungssteigerung des Propagationsmanagers

In diesem und im nächsten Kapitel wird die Leistungssteigerung des Propagationsmanagers diskutiert. Im Folgenden werden zunächst die im Rahmen dieser Arbeit durchgeführten Optimierungen vorgestellt, die zur Leistungssteigerung des Propagationsmanagers beigetragen haben. Die einzelnen Optimierungen werden anschließend anhand der in Kapitel 5 definierten Testfälle bewertet und schließlich wird die Leistungssteigerung des Gesamtsystems analysiert.

6.1 Maßnahmen zur Reduzierung der Bearbeitungszeit

Die Maßnahmen zur Verbesserung der Bearbeitungszeit des Propagationsmanagers können in zwei Gruppen aufgeteilt werden:

- Optimierungen am Propagationsmanager, um die einzelnen Bestandteile der Bearbeitungszeit zu reduzieren.

- Verwendung neuer leistungsstärkerer Komponenten für den Propagationsmanager. Das kann zum Bespiel eine andere Implementierung des XSLT-Prozessors für den Transformer oder ein anderer JMS-Server für den Queue Manager sein.
  Um solche Verbesserungsmaßnahmen am Propagationsmanager durchführen zu können, werden Vergleichsdaten für die Leistung der neuen Komponenten benötigt.

Im Folgenden werden Optimierungen am bestehenden System betrachtet, die zur Leistungssteigerung des Propagationsmanagers beitragen. Leistungssteigerungen unter Verwendung neuer Komponenten werden im Rahmen dieser Arbeit nicht behandelt.

Durch Analyse des Konzepts und der Implementierung des Propagationsmanagers konnten Optimierungsmaßnahmen getroffen werden, die zur wesentlichen Verbesserung der folgenden Bestandteile der Bearbeitungszeit führen:

- die Initialisierungs- und Beendigungszeiten der Prozessinstanzen,
- die Parsezeit einzelner Prozessdefinitionen,
- die Transformationszeiten und

- die Repository-Abfragenzeiten.

6.2 Optimierungen der Prozesskonnektoren

Die im Kapitel 5 durchgeführten Messungen haben gezeigt, dass die Initialisierung und die Beendigung der Prozessinstanzen sehr teuere Operation sind. Um diese Vorgänge zu optimieren, wurden die Stellen im Source Code analysiert, die diese Kosten verursachen. Schließlich gestellte sich herraus, dass ein großer Anteil dieser Kosten durch die so genannten *Prozesskonnektoren* verursacht wird.

Die Prozesskonnektoren sind für den Empfang, Versand, die Zwischenspeicherung und die Bestätigung von Nachrichten verantwortlich. Sie stellen die Verbindung der Prozessinstanzen mit der Messaging Middleware her (siehe Abbildung 6.1). Bei der Initialisierung einer Prozessinstanz wird für sie ein Prozesskonnektor initialisiert. Er registriert eine *Persistent Subscription* auf die Eingangs-Queue (eigentlich handelt es sich hier um einen Topic), so dass die Prozessinstanz alle ankommenden Nachrichten empfangen kann.
Damit eine Prozessinstanz Nachrichten an die Ausgangs-Queues des Queue Managers senden kann, wird bei der Initialisierung der Prozessinstanz ein *Session*-Objekt für die Queues initialisiert.

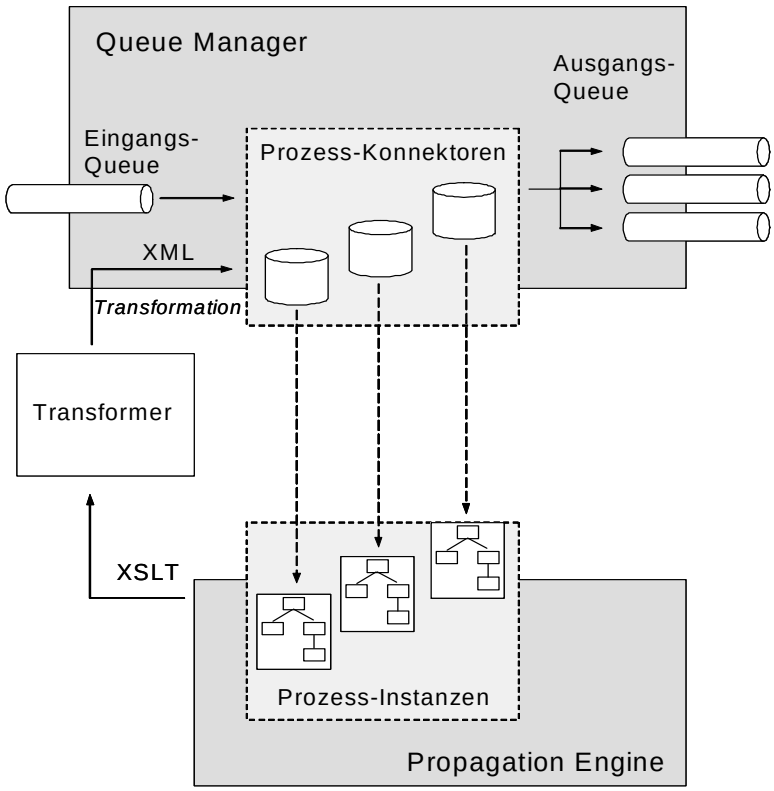

Abbildung 6.1: Prozess-Instanzen und Prozess-Konnektoren [KO01]

Insgesamt sind es drei Methoden, die den hohen Zeitaufwand verursachen: die Methode *createProcessConnector()* der Klasse *QueueManager* und die Methoden *start()* und *stop()* der Klasse *ProzessConnector*.

6.2.1 Durchzuführende Änderungen

Die vereinfachten Diagramme der Klassen *ProcessConnector* und *QueueManager* sind in Abbildung 6.2 dargestellt. Um die Initialisierungs- und Beendigungskosten der Prozessinstanzen zu reduzieren, wurde die Implementierung der Prozesskonnektoren analysiert. Folgendes wurde festgestellt:

- Bei direkter Übergabe (siehe Kapitel 2) werden die eingehenden Nachrichten durch die Propagationsengine im Hauptspeicher gespeichert und sobald die Prozessinstanz gestartet ist, wird die Nachricht an die Prozessinstanz übergeben. Aus diesem Grund wird bei direkter Nachrichtenübergabe überhaupt kein Subscriber benötigt. Neben der direkten Nachrichtenübergabe ist noch ein weiterer Modus zur Nachrichtenübergabe vorgesehen. Hierbei sollen Nachrichten mit Hilfe eines weiteren Topic-Objekts an die Prozessinstanzen übergeben werden (siehe Kapitel 2). Um diesen Modus umzusetzen, sollte man einen nicht persistenten Subscriber anstelle des Durable-Subscribers benutzen. Bei einer nicht persistenten Subscription können nur die Nachrichten empfangen werden, die nach der Initialisierung dieser Subscription in die Eingabe-Queue eingetroffen sind. Bei einer persistenten Subscription bleiben die Nachrichten in der Eingabe-Queue solange bis sie durch den persisten Subscriber bestätigt sind.

- Die Klasse *ProcessConnector* wird auch vom Propagationsmanager verwendet, um die eingehenden Nachrichten zu empfangen. Für das Empfangen der Nachrichten ist eine persistent Subscription auf die Eingabe-Queue des Queue Managers notwendig, damit die Nachrichten bei einem Ausfall des Propagationsmanagers nicht verloren gehen.

- Bei der Initialisierung eines ProzessConnectors (durch den Aufruf der Methode *createProzessConnector()* der Klasse *QueueManager*) wird das Session *outputQueueSession* initialisiert. Dieses Session-Objekt wird verwendet, um JMS-Nachrichten an die Ausgabe-Queues des Queue Managers zu senden. Bei Ausführung einer Propagate-Operation werden die durch den Propagationsmanager transformierten Datenänderungen in eine JMS-Nachricht gepackt und an die Ausgabe-Queues des Queue Managers gesendet.
  Um die Kosten zur Initialisierung des Session-Objekts zu reduzieren, kann man stattdessen einen Session-Pool verwenden. Der Session-Pool hat die Aufgabe mehrere bereits initialisierte Session-Objekte zu verwalten. Bevor eine Prozessinstanz eine Nachricht an die Zielsysteme senden will, holt sie ein Session-Objekt aus dem Session-Pool. Nach dem Senden der JMS-Nachricht wird das Session-Objekt in den Pool zurückgegeben. Durch die Wiederverwendung der Session-Objekte des Session-Pools werden die Initialisierungskosten für die Session-Objekte reduziert.

  Es gibt noch einen zweiten Grund für die Einführung des Session-Pools. Das Senden der transformierten Datenänderungen (ausgelöst durch die Propagate-Operation) an die Zielsysteme erfolgt mit Hilfe der Session *outputQueueSession*. Falls eine Prozessinstanz mehrere parallele Propagate-Operationen enthält, wird auf dieses Session-Objekt konkurrierend zugegriffen.

Die konkurrierende Verwendung von Session-Objekten ist laut der JMS-Spezifikation [SUN01] nicht erlaubt. Aus diesem Grund müssen die Zugriffe auf das Session-Objekt innerhalb einer Instanz der Klasse *ProcessConnector* synchronisiert werden.

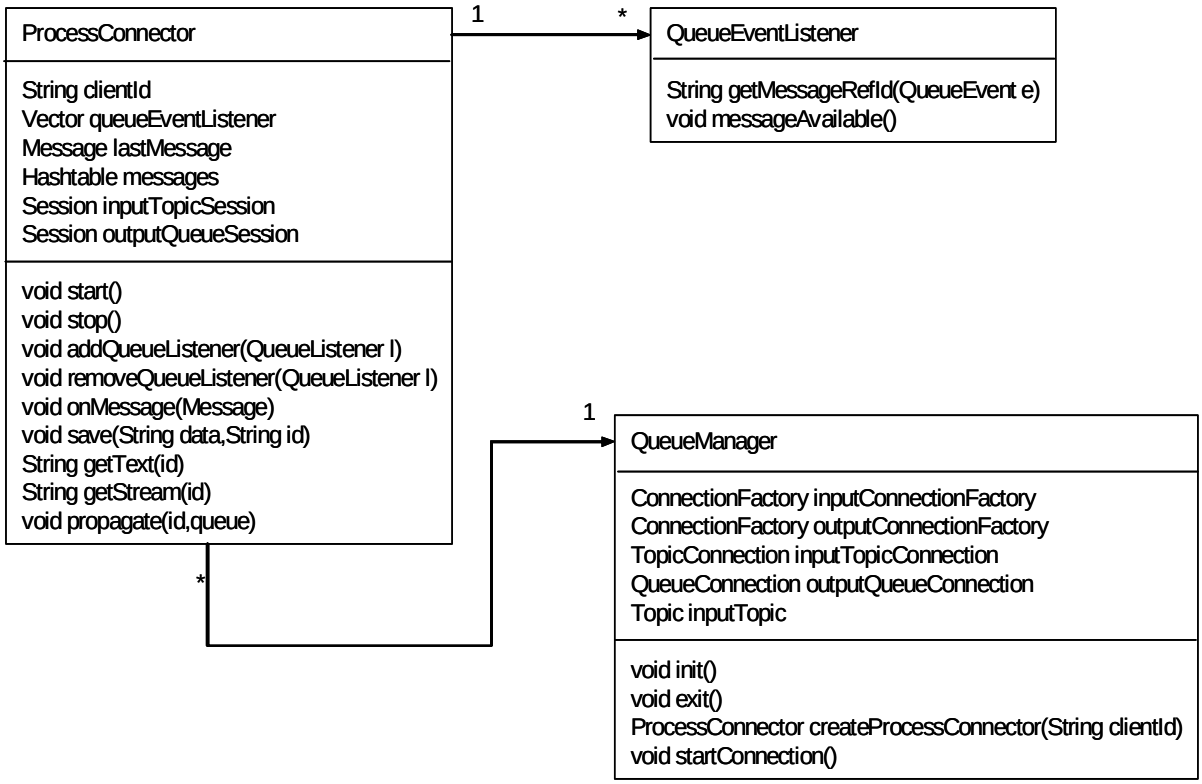

Abbildung 6.2: Diagramm der Klassen *ProcessConnector* und *QueueManager* [KO01]

6.2.2 Umsetzung

Um den ersten zwei Punkten gerecht zu werden, wird die Klasse *ProcessConnector* in zwei weitere Klassen gespalten. Die neue Klasse *ProcessConnector* wird nur von den Prozessinstanzen verwendet, um Nachrichten von der Propagationsengine zu bekommen oder Datenänderungen an die Zielsysteme zu propagieren (siehe Abbildung 6.3).

Die Klasse *PMConnector* hat die Aufgabe, die in die Eingangs-Queue eingehenden Nachrichten an die Propagationsengine zu senden. Beim Start des Propagationsmanagers wird eine Instanz dieser Klasse durch Aufruf der Methode *createPMConnector()* der Klasse *QueueManager* initialisiert. Nach dem Aufruf der Methode *start()* der Klasse *PMConnector* wird das Session-Objekt *inputTopicSession* initialisiert und die Propagationsengine ist in der Lage Nachrichten von Queue Manager zu empfangen.

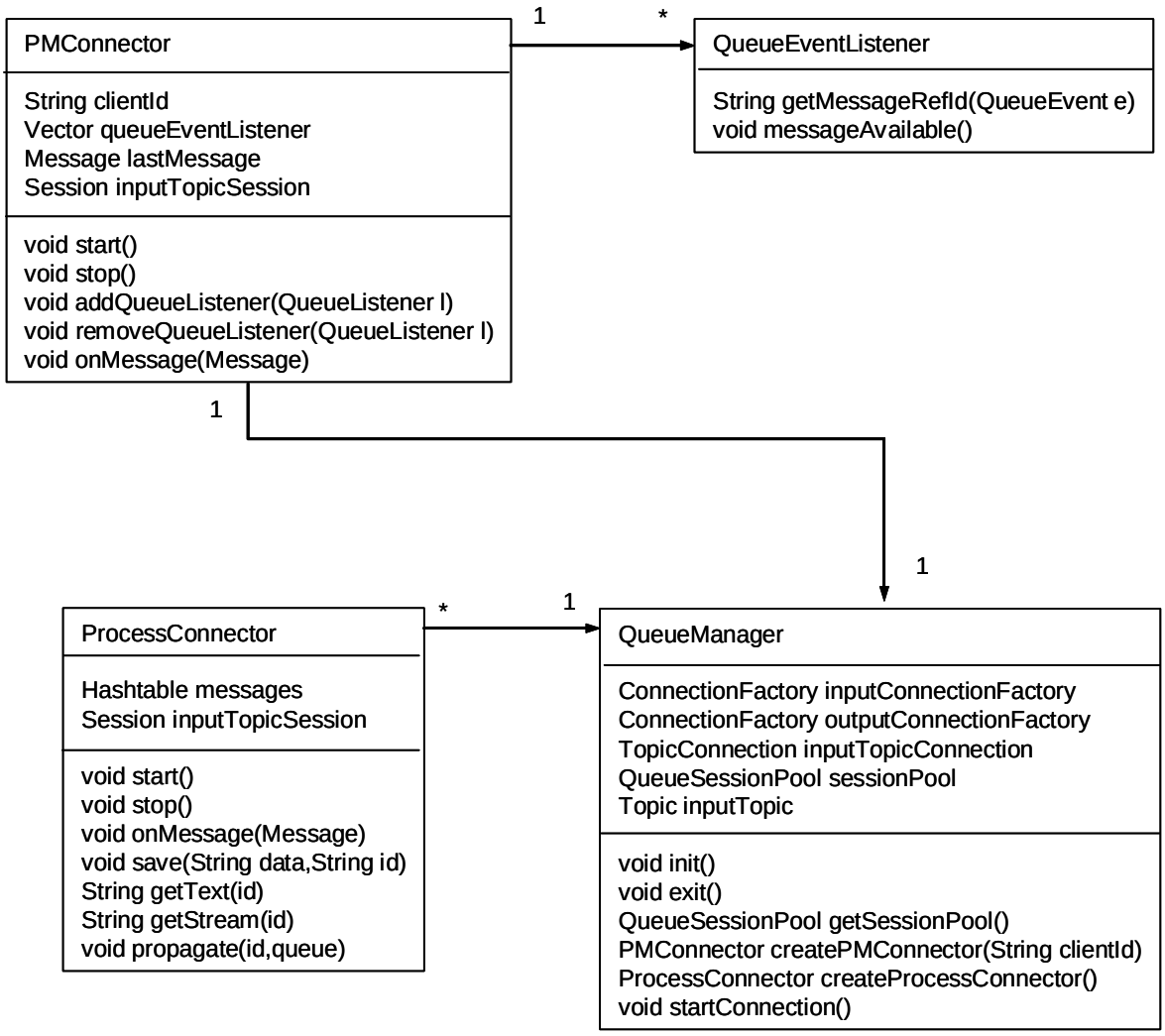

Abbildung 6.3: Diagrammen der Klassen *ProcessConnector* und *PMConnector* (neu)

Bei direkter Nachrichtenübergabe führen die Methoden *start()* und *stop()* der Klasse *ProcessConnector* keine Operationen aus. Wenn die Nachrichtenübergabe über eine weitere Queue des Queue Managers stattfinden soll, wird durch die Methode *start()* das Session-Objekt *inputTopicSession* initialisiert und durch die Methode *stop()* dieses Session-Objekt zerstört.

Aufgrund der oben diskutierten Änderungen, sollte auch die Klasse *QueueManager* angepasst werden. Um eine Instanz der Klasse *ProcessConnector* zu erzeugen, sollte man wie üblich die Methode *createProcessConnector()* aufrufen. Der Aufruf der Methode *createPMConnector()* initialisiert die Verbindung zwischen der Propagationsengine und dem Queue Manager.

6.2.3 Leistung des Propagationsmanagers nach den Änderungen

Um die durchgeführten Optimierungen an den Prozesskonnektoren zu bewerten, wurden Messungen mit Testfall 3 durchgeführt, um den Initialisierungs- und Beendigungsaufwand der Prozessinstanzen zu untersuchen. In Abbildung 6.4 sind die Messergebnisse für die Initialisierungszeiten der Prozessinstanzen vor den Änderungen und nach den Änderungen bei direkter Nachrichtenübergabe sowie bei Nachrichtenübergabe mit Hilfe eines weiteren Topics dargestellt. Das Entfernen des Durable-Subscribers hat den wesentlichen Anteil der Zeitaufwand reduziert. Die Einführung des Session Pools hat zu einer Verbesserung von 40 ms beigetragen.

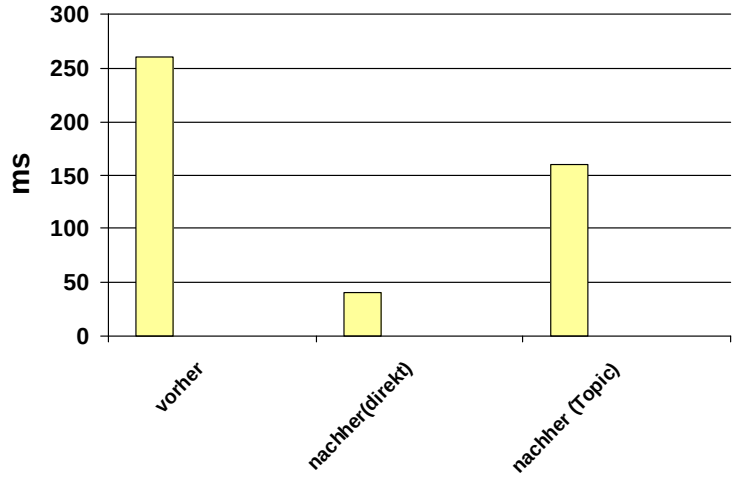

Abbildung 6.4: Initialisierungsaufwand der Prozessinstanzen

Die Beendigungszeiten der Prozessinstanzen sind in Abbildung 6.5 dargestellt. Vor den Änderungen benötigte eine Prozessinstanz 120ms, um die Persisten Subscription zu beenden. Die Beendigung einer Prozessinstanz, die Nachrichten direkt von der Propagationsengine bekommt (direkte Nachrichtenübergabe), dauert unter 10ms und bei Nachrichtenübergabe mit Hilfe eines weiteren Topics 30ms.

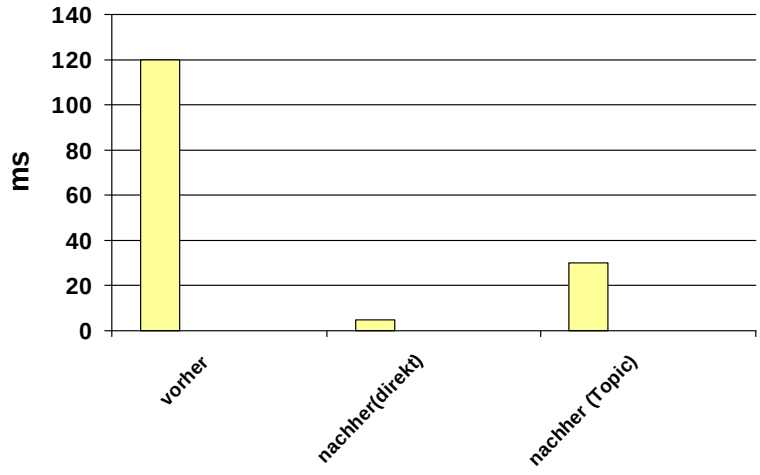

Abbildung 6.5: Beendigungsaufwand der Prozessinstanzen

6.3 Caching der Prozessdefinitionen

Während jeder Initialisierungsphase einer Prozessinstanz wird ein XRL+-Dokument geparst. Die Wahrscheinlichkeit, dass dasselbe XRL+-Dokument mehrmals geparst wird, hängt von der Wahrscheinlichkeit ab, mit der das dem XRL+-Dokument entsprechende Quellsystem Datenänderungen produziert und diese mit Hilfe des Propagationsmanagers propagiert. Die durchgeführten Messergebnisse haben gezeigt, dass gerade das Parsen großer Abhängigkeitsskripte (siehe die Ergebnisse von Testfall 3 in Kapitel 5) mit einem hohen Zeitaufwand verbunden ist. Als groß werden die XRL+-Dokumente bezeichnet, die mehr als zehn Einträge enthalten. Das Parsen des XRL+-Dokuments, das für Testfall 3 verwendet wird, das zehn Einträge enthält, dauert laut die Ergebnisse von Testfall 4 45ms, bei Dokumenten mit 20 Einträgen 70ms. Durch Einführung von Caching sollte das wiederholte Parsen von XRL+-Dokumenten reduziert werden.

Durch das Caching der Prozessdefinitionen erwartet man zum einen, dass die Kosten zum Parsen der XRL+-Dokumente reduziert werden. Zum anderen werden dadurch auch die Repository-Zugriffe reduziert.

6.3.1 Der XRL+-Parser

Das Einlesen der XRL+-Dokumente erfolgt durch einen XML-Parser. Es existieren zwei verschiedene Typen von XML-Parsern: SAX-Parser und DOM-Parser.
Der SAX-Parser ist ereignisgesteuert, d.h. ein Ereignis wird ausgelöst, wenn der Parser ein XML-Konstrukt erkennt. Die Anwendung, die den Parser verwendet, stellt Methoden zur Behandlung dieser Ereignisse bereit. Für das eingelesene XML-Dokument wird vom Parser keine Datenstruktur aufgebaut [SUN2].
Der DOM-Parser (DOM steht für Document Object Model) erzeugt während des Parsens eine Baumstruktur, bestehend aus Objekten, die jeweils ein XML-Konstrukt beschreiben. Das XML-Dokument wird dadurch vollständig im Hauptspeicher nachgebildet.

Zum Parsen der XRL+-Dokumente wird ein SAX-Parser verwendet. Beim Einlesen eines XRL+-Dokument wird eine dem DOM ähnliche Baumstruktur erzeugt. Diese Baumstruktur ist die ausführbare Ausprägung der Prozessdefinition, die Prozessinstanz.

6.3.2 Funktionsweise des Caches

Um wiederholtes Parsen eines XRL+-Skripts zu vermeiden, wird die bereits erzeugte Prozessinstanz im Hauptspeicher gespeichert und seine Datenstruktur mehrmals verwendet. Die Datenstruktur einer Prozessinstanz enthält sowohl die Daten vom XRL+-Dokument als auch Zustandsinformation der ausführbaren Prozessinstanz. Die Zustandsinformation in der Datenstruktur der Prozessinstanz (für diese Zwecke wäre ein DOM-Baum wahrscheinlich eine bessere Lösung) bringt zwei Probleme mit sich:

- Eine Prozessinstanz kann ohne weiteres nicht mehr als einmal verwendet werden.
- Eine Prozessinstanz ist nicht Threadsafe, d.h. auf dieselbe Prozessinstanz darf nicht mehr als ein Thread gleichzeitig zugreifen.

Um diese zwei Probleme zu lösen, werden Kopien der im Cache gespeicherten Prozessinstanz erzeugt und diese Kopien werden zum Ausführen verwendet. Beim Erzeugen einer neuen Kopie einer Prozessinstanz werden nur die Daten des XRL+-Dokuments kopiert, die Zustandsinformation der Prozessinstanz nicht.

Die Einführung dieses Caches führt zu folgenden Änderungen des Ablaufs der Propagationsengine:

- Beim Eintreffen einer neuen Nachricht in die Eingabe Queue wird die Nachricht an die Propagationsengine übergeben. Die Propagationsengine durchsucht zunächst den Cache nach der gesuchten Prozessinstanz.

- Falls die gesuchte Prozessinstanz im Cache ist, wird eine Kopie von ihr der Propagationsengine zurückgegeben.

- Falls die Prozessinstanz nicht im Cache ist, wird das entsprechende XRL+-Dokument aus dem_Repository geholt. Das Dokument wird geparst und eine Prozessinstanz erzeugt. Diese wird im Cache gespeichert und eine Kopie der Instanz wird der Propagationsengine zurückgegeben.

6.3.3 Kapazität und Ersetzungsstrategie

Zwei weitere Anforderungen, die bei der Implementierung des Caches berücksichtigt werden müssen, sind die Kapazität des Caches und seine Ersetzungsstrategie. Die Kapazität des Caches definiert die maximale Anzahl der Einträge der gespeicherten Prozessinstanzen im Cache. Diese Größe soll konfigurierbar sein, da es hauptsächlich von der Größe des verfügbaren Hauptspeichers abhängig ist, wie viele Prozessinstanzen im Cache gespeichert werden können.

Wenn eine neue Prozessinstanz im Cache gespeichert werden soll und dort kein Platz mehr vorhanden ist, muss eine andere gespeicherte Prozessinstanz entfernt und die neue an ihren Platz gespeichert werden. Welcher Eintrag vom Cache entfernt werden soll, hängt von der Ersetzungsstrategie des Caches ab. Es gibt viele Heuristiken, die für diese Zwecke verwendet werden. Einige von ihnen werden hier kurz erwähnt [LK02]:

- *FIFO* (First In First Out): Die Annahme bei dieser Strategie ist, dass der am längsten im Hauptspeicher liegende Eintrag in der Zukunft nicht mehr benötigt wird.

- *LRU* (Least Recently Used): Der am längsten nicht mehr verwendete Eintrag im Cache wird ersetzt. Diese Strategie basiert auf die Annahme, dass der am längsten nicht verwendeten Cacheeintrag auch in der Zukunft selten verwendet wird.

- *LFU* (Least Frequently Used): Der am wenigsten benutzte Eintrag im Cache, wird ersetzt.

Da es keine Informationen über die Wahrscheinlichkeit gibt, mit der auf die XRL+-Dokumente zugegriffen wird, scheint die LRU-Strategie für die Anforderungen des Caches gut geeignet zu sein.

6.3.4 Messergebnisse für das Caching der Prozessinstanzen

Um das Caching der Prozessdefinitionen zu bewerten, wurden Messungen mit dem Testfall 3 durchgeführt. Die Messergebnisse sind in Abbildung 6.6 dargestellt. Der Zeitaufwand, der durch das Parsen der XRL+-Dokumente verursacht wird, wird mit Hilfe des Caching der Prozessdefinitionen deutlich reduziert.

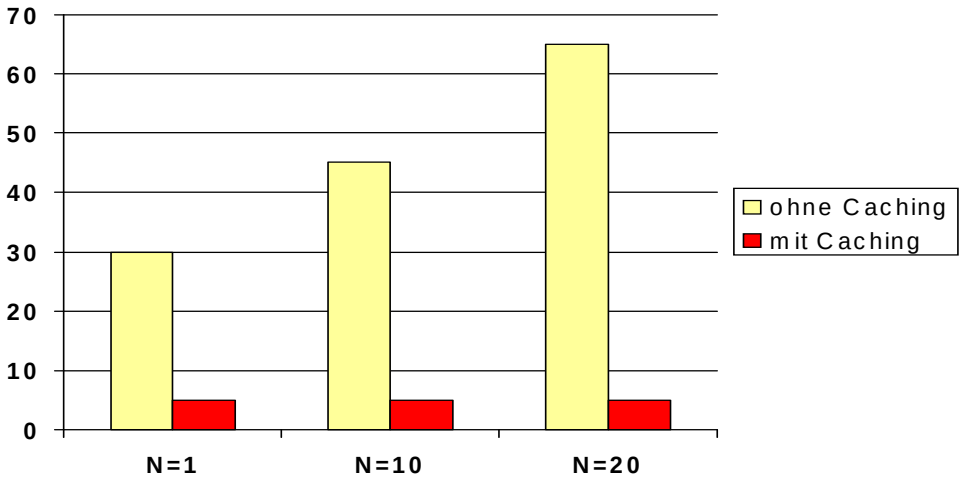

Abbildung 6.6: Leistungsverbesserung durch Caching der Prozessdefinitionen

6.3.5 Entwurf und Implementierung

Damit Kopien einer Prozessinstanz erzeugen werden können, wird, wie in Abbildung 6.7 dargestellt ist, die Klasse *RoutingElement* um die Methode *createNewCopy()* erweitert. Diese neue Methode ist als *abstract* definiert und muss aus diesem Grund durch alle Klassen in der Klassenhierarchie implementiert werden.

Durch den Aufruf dieser Methode wird eine neue Kopie der Instanz erzeugt. Bei diesem Vorgang werden alle untergeordneten Elemente der Instanz rekursiv durchgegangen und deren Daten kopiert. Auf diese Weise wird eine neue Kopie der Prozessinstanz erzeugt, die exakt die gleichen XRL+-Daten enthält wie die ursprüngliche Prozessinstanz.

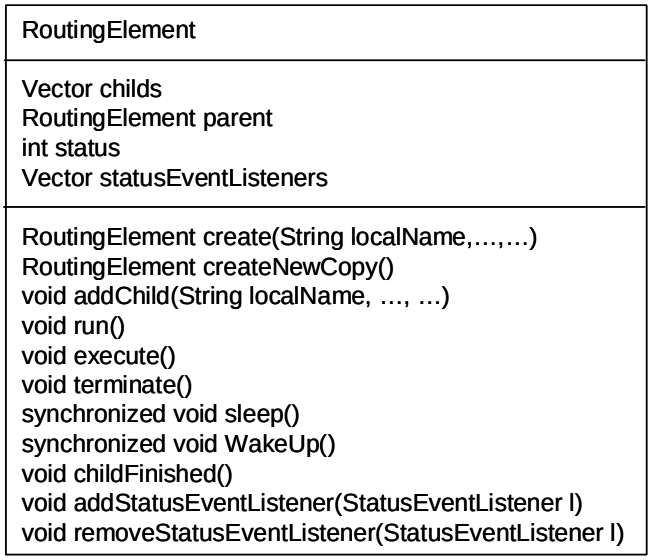

Abbildung 6.7: Die Klasse RoutingElement geändert (neu)

Die Klasse *ProcessInstanceCache* (siehe Abbildung 6.8) implementiert das Caching der Prozessinstanzen. Die Kapazität wird bei der Initialisierung des Caches durch den Aufruf des Konstruktors *ProcessInstanceCache(int size)* angegeben. Durch die Methode *flushAll()* können alle Einträge des Cache entfernt werden. Einzelne Einträge können durch Aufruf der Methode *flush()* entfernt werden. Die Methode *getInstance()* gibt eine ausführbare Instanz der Klasse *ProcessInstance* zurück, die von der Propagationsengine verwendet werden kann. Die Kapazität des Caches kann durch die Methode *getSize()* abgefragt werden.

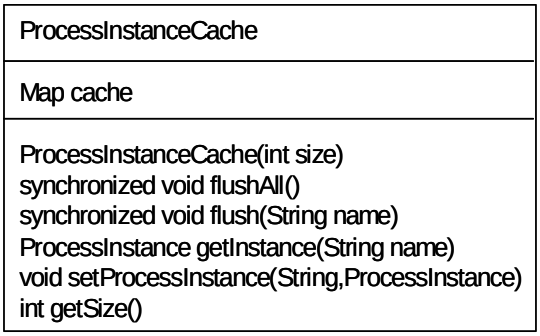

Abbildung 6.8: Die Klasse ProcessInstanceCache

6.4 Stylesheet Cache

Die nächste Erweiterung des Propagationsmanagers verfolgt das Ziel die Transformationszeiten und die Repository-Abfragezeit für die Transformationsskripte durch die Einführung eines weiteren Caches im System zu reduzieren. Die Idee verfolgt den in [XSL03] beschriebenen Ansatz.

Bevor ein XSLT-Skript ausgeführt werden kann, wird, wie in jeder Programmiersprache, den Source Code in ein internes Format umgewandelt. Den gleichen Vorgang kann man bei den Compilern finden. Der XSLT-Prozessor liest das

XSLT-Dokument ein, analysiert den Code, wandelt ihn in einen Zwischencode um und transformiert damit die XML-Daten. Wird dieses Vorgehen in Bezug auf die Leistung analysiert, kann festgestellt werden, dass für die selten geänderten XSLT-Dokumente diese Situation sehr „ungünstig“ ist. Jedes Mal, wenn dieselbe XSLT-Transformation durchgeführt wird, konvertiert der XSLT-Prozessor den Source Code in einen Zwischencode.
Genau diese Situation ist im Propagationsmanager vorhanden. Die im Repository gespeicherten XSLT-Dokumente werden oft verwendet, aber selten geändert. Die Idee des neuen Caching-Ansatzes verfolgt das Ziel, die XSLT-Dokumente nur einmal zu parsen und den erzeugten Zwischencode für häufige Verwendung in den Hauptspeicher, in den so genannten *StylesheetCache*, zu speichern.

Die Anforderungen an den neuen Cache bleiben die gleichen wie die beim Caching der Prozessinstanzen. Die Kapazität des neuen Caches soll konfigurierbar sein und die Ersetzungsstrategie bleibt LRU (least recently used).

Neben diesen Anforderungen haben sich noch weitere Anforderungen an den Stylesheet-Cache und an den Transformer ergeben (diese wurde im Laufe der Arbeit festgestellt). Es gibt zurzeit unterschiedliche XSLT-Prozessoren, die den Zwischencode der Transformationsskripte in unterschiedlicher Weise generieren und für denselben Vorgang unterschiedliche Zeiten benötigen. Falls die Kapazität des Stylesheet-Cache zu klein gewählt wird, kann es vorkommen, dass die Kosten zur erneuten Erzeugung von Zwischencodes den Gewinn durch die Einführung des Caches übersteigen. Aus diesem Grund sollte man in der Lage sein, den Stylesheet-Cache auszuschalten. Damit der Transformer sowohl mit dem Cache als auch ohne ihn arbeiten kann, sollte seine Funktionalität an diese Anforderung angepasst werden.

6.4.1 Templates API oder Transformer Klasse

Es gibt generell zwei Möglichkeiten den Stylesheet-Cache zu realisieren: mit Hilfe der Klasse *Transformer* oder mit Hilfe des *Templates API* (siehe Abbildung 6.9).

Die Klasse, *Transformer*, wird verwendet, um XSLT-Transformationen durchzuführen. Jede Instanz dieser Klasse enthält Zustandsinformationen und kann aus diesem Grund nicht konkurrierend verwendet werden.

Die zweite Möglichkeit ist das *Templates API*. Die verschiedenen XSLT-Prozessoren implementieren das *javax.xml.transform.Templates* Interface unterschiedlich. Eine Templates-Instanz repräsentiert ein ausführbares (engl. runtime) XSLT-Dokument. Ob das XSLT-Dokument kompiliert oder nur in eine optimierte Darstellung umgewandelt wird, hängt vom entsprechenden XSLT-Prozessor ab.
Aus einer Templates-Instanz kann eine Transformer-Instanz erzeugt werden und sie dann für die Tranformation verwendet. Der Vorteil einer Templates-Instanz gegenüber einer Transformer-Instanz ist es, dass sie konkurrierend verwendet werden kann.

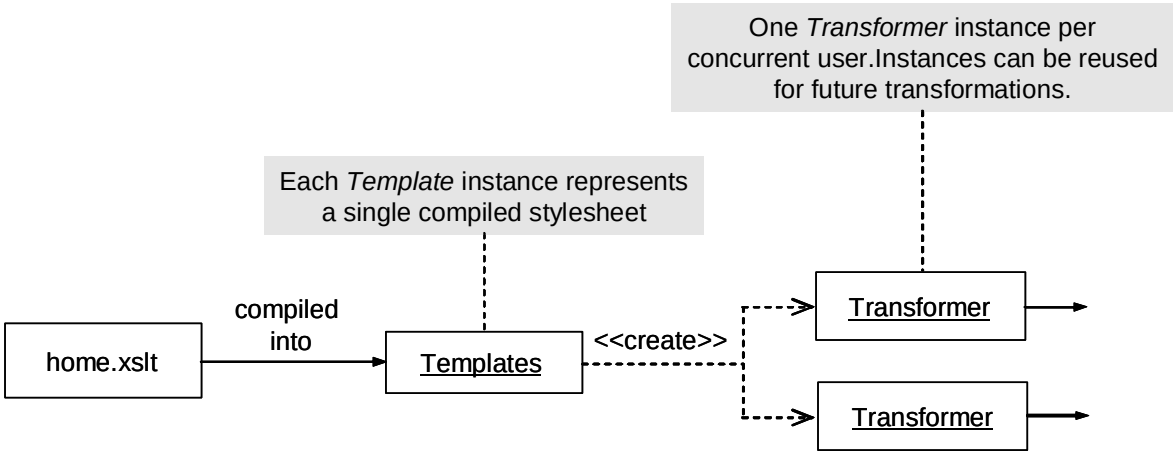

Abbildung 6.9: Beziehung zwischen Templates und Transformer

Abbildung 6.10a) zeigt den geänderten Ablauf bei Durchführung von Transformationen. Der Transformer durchsucht den StylesheetCache nach dem entsprechenden, kompilierten XSLT-Dokument. Falls die Daten dort nicht vorhanden sind, wird das XSLT-Dokument vom Repository geholt. Anschließend wird das XSLT-Dokument kompiliert und dem Transformer übergeben.

Falls der Cache abgeschaltet wird, holt sich der Transformer die Daten, wie in Abbildung 6.10b) gezeigt, direkt vom Repository.

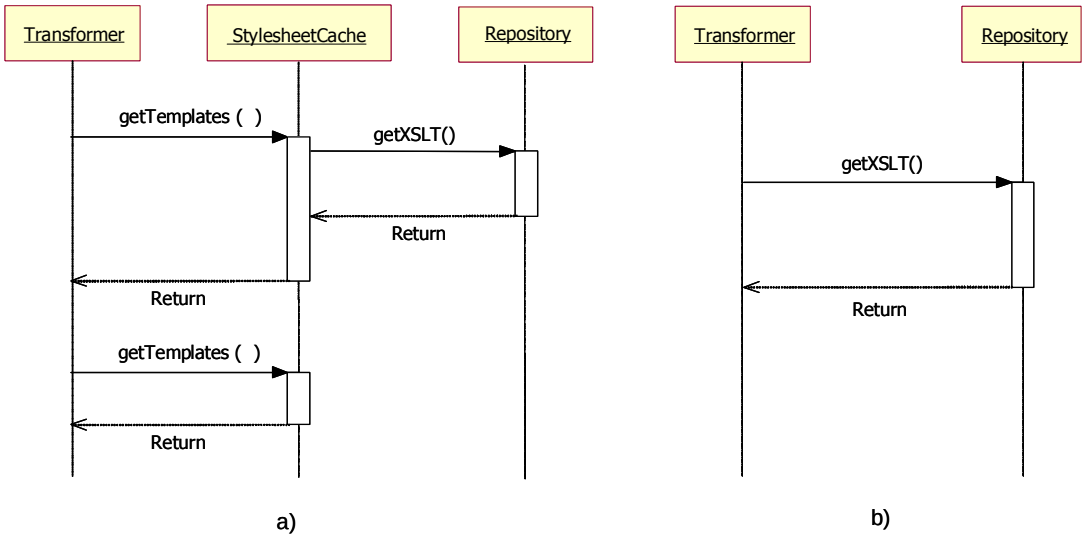

Abbildung 6.10: Funktionsweise des Stylesheet Cache

6.4.2 Entwurf und Implementierung

Das Templates-Interface ist in Abbildung 6.11 dargestellt. Zur Durchführung einer Transformation wird zunächst die Methode *newTransformer()* aufgerufen, um eine

Transformer-Instanz zu erzeugen. Mit Hilfe dieser *Transformer*-Instanz kann die Transformation der XML-Daten durchgeführt werden.

```
public interface Templates {
   java.util.Properties getOutputProperties( );
   java.xml.transform.Transformer newTransformer( )
            throws TransformerConfigurationException;
}
```

Abbildung 6.11: Das Templates-Interface

Die Methode *getTemplates()* der Klasse *StylesheetCache*, in Abbildung 6.12 dargestellt, liefert eine Instanz der Klasse *Templates*, mit der die Transformation der XML-Daten durchgeführt werden können.

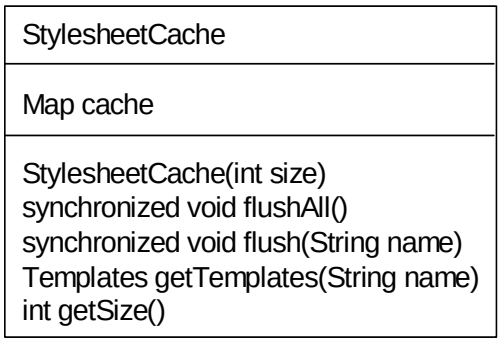

Abbildung 6.12: Die Klasse StylesheetCache

Für die Transformation der XML-Daten wird die Methode *transform()* der Klasse *XSLTProcessor* aufgerufen. Neben der Implementierung der Klasse *StylesheetCache* wird die Methode *transform()* der Klasse *XSLTProcessor* angepasst, um die Arbeit des Transformers mit und ohne Cache zu gewährleisten.

6.4.3 Vergleich der Leistung des Transformers mit und ohne StylesheetCache

Um die Leistung des Transformers mit Stylesheet-Cache bewerten zu können, wurde Testfall 2 durchgeführt. Die Messergebnisse sind in Tabelle 6.1 dargestellt. Für die Nachrichten mit wenigen Einträgen (Nachricht 1) zeigen die Messdaten eine deutliche Leistungssteigerung des Transformers. Die Leistungssteigerung für die Nachrichten mit vielen Einträgen (Nachricht 2 und Nachricht 3) ist dagegen geringer.
Diese Ergebnisse können folgendermaßen erklärt werden. Die Transformationszeit ohne Stylesheet-Cache besteht aus der Zeit zur Erzeugung des Zwischencode und der „reine“ Transformationszeit. Das Erzeugen des Zwischencode eines XSLT-Dokuments beansprucht immer denselben Zeitaufwand. Für die kleinen Nachrichten ist dieser Zeitaufwand offenbar vergleichbar hoch im Vergleich mit der „reinen“ Transformationszeit. Bei den Transformationen mit Stylesheet-Cache wird dagegen den Zwischencode nur einmal generiert.

Testfall	Anzahl der Einträge in der Eingabe-Nachricht	Transformationszeit	
		Vorher	Nachher
Testfall 2.1	1	90ms	40ms
Testfall 2.1	10	100ms	70ms
Testfall 2.1	100	100ms	80ms
Testfall 2.2	1	70ms	40ms
Testfall 2.2	10	110ms	60ms
Testfall 2.2	100	140ms	80ms
Testfall 2.3	1	90ms	30ms
Testfall 2.3	10	100ms	50ms
Testfall 2.3	100	140ms	80ms

Tabelle 6.1: Ergebnisse mit StylesheetCache für den Testfall 2

6.5 Auffrischung der eingeführten Caches

Die Einführung von Caching im Propagationsmanager, um seine Leistung zu optimieren, bringt ein anderes Problem mit sich. Während des Betriebs des Propagationsmanagers kann ein Propagations- oder Transformationsskript mit Hilfe des Abhängigkeitsmanagers editiert und schließlich geändert werden. Ob solche Änderungen erlaubt werden sollen, ist noch zu diskutierten, da sie auch die laufende Ausführung einer Prozessinstanz beeinflussen können.
Wenn das geänderte Skript nicht in einem der Caches geladen ist, wird diese Änderung keine Probleme bereiten. Falls das Skript in einem der Caches bereits geladen wurde, unterscheidet sich sein Inhalt von diesem im Repository. Eine Auffrischung des entsprechenden Caches oder zumindest des entsprechenden Eintrag im Cache ist daher notwendig. Zurzeit kann eine Auffrischung nur durch einen Restart des Propagationsmanagers erzwungen werden. Ob diese Lösung für die Anforderungen des SIES in Hinblick auf das oben genannte Problem ausreichend ist, ist noch zu diskutieren.

6.6 Die Leistung des Propagationsmanager nach den Optimierungen

Um die Leistungssteigerung des Propagationsmanagers nach den Optimierungen zu untersuchen, wurden die Testfälle 4, 5 und 6 durchgeführt. Die Messergebnisse dieser Testfälle sind in Tabelle 6.2, Tabelle 6.3 und Tabelle 6.4 dargestellt (in Klammern stehen die Messergebnisse, die vor den Optimierungen erhoben wurden). Alle Messungen wurden mit dem direkten Nachrichtenübergabe-Modus durchgeführt.

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100 (100)	100 (100)	311 (902)	100 (100)
Auslastung	0.34 (0.88)	0.27 (0.23)	0.12 (0.20)	0.43 (0.27)
Kapazität	5.25 (1.18)	6.66 (4.44)	46.2 (46.0)	4.13 (3.76)
Bearbeitungszeit	19s (84s)	15s (22.5s)	6.7s (19.6s)	24s (26.6s)
Gesamtzeit	55.8s (vorher: 95.4s)			

Tabelle 6.2: Messergebnisse für den Testfall 4

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100 (100)	600 (600)	304 (2550)	100 (100)
Auslastung	0.39 (0.86)	0.71 (0.48)	0.12 (0.44)	0.49(0.15)
Kapazität	4.5 (0.70)	14.9 (7.5)	43 (34.7)	3.62 (4.00)
Bearbeitungszeit	22s (141s)	40.3s (79.1s)	7s (73s)	27s (25s)
Gesamtzeit	56.7s (vorher: 164.2s)			

Tabelle 6.3: Messergebnisse für den Testfall 5

	Propagationsengine	Transformer	Repository	Queue Manager (Propagation)
Aufträge	100 (100)	300 (300)	305 (1674)	300 (300)
Auslastung	0.31 (0.81)	0.34 (0.42)	0.12 (0.48)	0.70 (0.57)
Kapazität	4.3 (0.91)	11.5 (5.23)	34 (26)	5.7 (3.94)
Bearbeitungszeit	23s (108.9s)	26s (57.3s)	8.9s (63.7s)	52s (76s)
Gesamtzeit	74.7s (vorher: 134.5s)			

Tabelle 6.4: Messergebnisse für den Testfall 6

6.7 Mögliche Engpässe im Propagationsmanager

Im letzten Abschnitt wurden die Messergebnisse für den optimierten Propagationsmanager dargestellt. Diese Messungen wurden mit einer „künstlichen“ Arbeitslast durchgeführt und weisen auf drei mögliche Engpässe hin, die bei einer produktiven Arbeitslast auftreten können:

- Die Anzahl der durchzuführenden Transformationen und deren Komplexität können den Transformer zur Engpasskomponente des Propagationsmanagers machen. Die Ergebnisse aus dem Testfall 5 bestätigen diese Behauptung. Bei diesem Testfall ist der Transformer, mit einer Auslastung von 71% (0.71) und

Gesamtbearbeitungszeit von 40.3s, die Komponente, die die Leistung des Propagationsmanagers bestimmt.

- Die Messergebnisse für den Testfall 6 zeigen, dass der Queue Manager, mit einer Auslastung von 70% (0.70) und Gesamtbearbeitungszeit von 52s, die Komponente ist, die die Leistung des Propagationsmanagers bestimmt. Somit kann der Queue Manager der Engpass im System werden, falls die produktive Arbeitslast viele Propagate-Operationen enthält.

- Bei den durchgeführten Messungen weist die Kapazität der Propagationsengine darauf hin, dass mit der verwendeten Systemkonfiguration theoretisch nicht mehr als vier Datenänderungspropagationen pro Sekunde durchgeführt werden können. Falls die Nachrichtenübergabe über eine weitere Queue erfolgt, wird dieser Wert deutlich niedriger sein.
  Die Kapazität der Propagationsengine muss unbedingt dann berücksichtigt werden, wenn der Propagationsmanager für einen höheren Durchsatz, als der Wert dieser Messgröße, ausgelegt werden soll.

Die Messergebnisse im letzten Abschnitt zeigen eine sehr geringe Auslastung des Repository nach der Umsetzung der Optimierungen. Aus diesem Grund kann man davon ausgehen, dass das Repository nicht zum Engpass werden kann, wenn es mit einem Propagationsmanager verwendet wird. Da das Repository mit Hilfe einer Datenbank umgesetzt wurde, muss die Leistung dieser Datenbank ständig überprüft werden. Falls das Repository viele Daten enthält, muss die Größe des *Buffer Pools* dieser Datenbank entsprechend erhöht werden. Der Buffer Pool ist ein Teil des Hauptspeichers, der von der Datenbank zur temporären Speicherung von Tabellen-, Index- und Katalogdaten verwendet wird [HT98].
Bei der Überwachung des Buffer Pools ist die *Hit Ratio* des Buffer Pools die wichtigste Größe. Sie bezeichnet den Prozentsatz der erfolgreichen Zugriffe auf den Buffer Pool. Falls dieser Wert bei der Datenbank im laufenden Betrieb unter 80% fällt, ist der Buffer Pool zu klein gewählt.

Kapitel 7

Weitere Konzepte

In diesem Kapitel werden verschiedene Konzepte und Ansätze zur Leistungssteigerung des Propagationsmanagers diskutiert, die im Rahmen dieser Arbeit nicht realisiert wurden.

7.1 Semantische Analyse für die Propagationsskripte

Die Propagationsskripte sind XML-Dokumente und werden mit Hilfe eines XML-Parsers eingelesen und analysiert. Hierbei beschränkt sich die Analyse auf die Überprüfung der XRL+-Schema auf die Daten der Propagationsskripte. Wie im Folgenden gezeigt wird, ist diese Analyse nicht in der Lage, gewisse Konflikte in den Propagationsskripten zu finden und Optimierungen in den XRL+-Dokumenten vorzunehmen, die zur Verbesserung der Ausführungszeit der Prozessinstanzen beitragen können. Für diese Zwecke ist das XRL+-Schema, das die Struktur der XRL+-Dokumente beschreibt, nicht ausreichend.

Viele der in Kapitel 5 durchgeführten Messungen am Propagationssystem hätten deutlich bessere Ergebnisse aufgewiesen, wenn die definierten Operationen in den Propagationsskripten analysiert wurden. Da es sich hier um eine Analyse handelt, bei der die XRL+-Dokumente auf gewisse Bedingungen überprüft werden, wird sie als semantische Analyse für die Propagationsskripte bezeichnet.

7.1.1 Problemstellung

Das Problem wird durch das Beispiel in Abbildung 7.1 verdeutlicht. Im dargestellten Propagationsskript gibt es drei Probleme, die im aktuellen Konzept des Propagations- und des Abhängigkeitsmanagers nicht berücksichtigt sind.

```
(1)   <?xml version="1.0" encoding="UTF-8"?>
(2)   <xrl:ROUTE id="1" xmlns:xrl="http://informatik.uni-stuttgart.de/XRL">
(3)       <xrl:SEQUENCE>
(4)           <xrl:WAIT sync="1">
(5)               <xrl:MESSAGE_EVENT initial="yes" schema="SCHEMA_IN" system="SYSTEM_IN“
(6)                 type="any" xml_out=" in_data"/>
(7)           </xrl:WAIT>
(8)           <xrl:TRANSFORM xml_in= "null_data" xml_out=„fun_data" xslt="xslt_0"/>
(9)           <xrl:PARALLEL sync="2">
(10)              <xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt_1"/>
(11)              <xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt_2"/>
(12)          </xrl:PARALLEL>
(13)          <xrl:PROPAGATE xml="out_data" system="SYSTEM_OUT" schema="SCHEMA_OUT"/>
(14)          <xrl:TRANSFORM xml_in="in_data" xml_out= "trash_data" xslt="xslt_3"/>
(15)      </xrl:SEQUENCE>
(16)  </xrl:ROUTE>
```

Abbildung 7.1: Raceconditions und überflüssige Operationen im XRL+-Dokument

Diese werden im Folgenden vorgestellt:

- Die Zeile (10) und (11) definieren zwei Transformationen, die parallel zueinander ausgeführt werden sollen. Die Ein- und die Ausgabedaten beider Transformationen werden durch die gleichen Attribute referenziert. Die Frage hier lautet: Welche Daten werden dann in Zeile (13) an das Zielsystem weitergeleitet?

  Das Konzept des Propagationssystems behandelt dieses Problem nicht. Damit man diese Frage beantworten kann, muss man die Implementierung des Propagations- und Abhängigkeitsmanagers anschauen. Das Ergebnis dieser zwei Operationen ist von der Reihenfolge der ausgeführten Transformationen abhängig. Wenn die erste Transformation zuerst fertig ist, kann sie ihr Ergebnis in *out_data* der temporären Daten der Prozessinstanz speichern. Die zweite Transformation kann diese Daten allerdings überschreiben. Da der Schreibvorgang der Ergebnisse in den temporären Bereich einer Prozessinstanz keine atomare Operation ist (die entsprechende Methode ist nicht als *synchronized* definiert), kann es auch vorkommen, dass beide Operationen gleichzeitig auf die Daten *out_data* schreiben. Das weist darauf hin, dass das Ergebnis der parallelen Ausführung beider Transformationen in Zeilen (10) und (11) nicht definiert ist.
  Transformationen oder Filterungsoperationen, die parallel zueinander ausgeführt werden und deren Ausgabedaten durch dieselben Attribute referenziert werden, sind Konfliktoperationen innerhalb des XRL+-Dokuments.

  Es gibt einige Möglichkeiten diese Konflikte aufzulösen. Die Erste besteht darin den Schreibvorgang zu synchronisieren. Eine weitere Alternative wäre, die Synchronisation der Konfliktoperationen durch Verwendung von Sperren zu gewährleisten. Eine dritte Möglichkeit wäre, die Speicherung von Propagationsskripten mit solchen Konflikten ins Repository zu verbieten oder diese Konfliktdaten mit einem Flag im Repository zu vermerken, damit sie durch den Propagationsmanager nicht ausgeführt werden. Der Administrator kann die vermerkten Dokumente zu einem späteren Zeitpunkt ändern und die Konfliktdaten in den Propagationsskripten entfernen.

Im Grunde genommen sind die ersten zwei Alternativen äquivalent. Sie führen zu einer seriellen Ausführungsreihenfolge der Konfliktoperationen. Leider sind beide Alternativen nicht deterministisch, da die Reihenfolge der Konfliktoperationen und dadurch auch das Ergebnis dieser Operationen nicht festgelegt sind. Aus diesem Grund sind sie für den Einsatz im Propagationssystem nicht geeignet.
Damit verbleibt nur die dritte Alternative.

- In Zeile (14) wird eine Transformation definiert, deren Ergebnis nicht mehr verwendet wird. Das bedeutet, dass diese Operation überflüssig ist, da ihr Ergebnis weder für eine Propagation noch als Eingabe für weitere Operationen verwendet wird. Dadurch entstehen unnötige, in machen Fällen sogar erhebliche Kosten bei der Ausführung eines Propagationsskripts.

  Außer dieser Klasse von überflüssigen Operationen gibt es noch weitere, die im vorgestellten Beispiel nicht dargestellt sind. Das sind Operationen, die mehr als einmal auf den gleichen Daten durchgeführt werden. Die Erkennung dieser Klasse von überflüssigen Operationen ist eine etwas komplexere Aufgabe. Um solche Operationen erkennen zu können, muss der Datenfluss im Propagationsskript analysiert werden.

  Die Erkennung überflüssiger Operationen vor und während der Ausführung einer Prozessinstanz, wäre eine Optimierungsmöglichkeit.

- Zeile (8) definiert eine Transformation, die nicht definierte Daten verwendet. Dieses Problem wird in der Implementierung des Propagationsmanagers berücksichtigt. Bevor eine Operation ausgeführt wird, wird überprüft, ob die benötigten Daten schon vorhanden sind. Falls ja wird die Operation ausgeführt, ansonsten wird sie abgebrochen.

7.1.2 Alternativen für die Umsetzung

Ein produktives System sollte in der Lage sein, die dargestellten Probleme aufzulösen und die entsprechenden Optimierungen durchzuführen. Um diese Probleme zu lösen, muss das bestehende Konzept des SIES um eine semantische Analyse erweitert werden.

Es gibt zwei Alternativen, zu welchem Zeitpunkt die semantische Analyse durchgeführt werden kann. Die erste Alternative wäre, die Analyse während jeder Prozessinitialisierung, nachdem die Prozessdefinition eingelesen wurde, durchzuführen. Das bedeutet, dass die gleiche Arbeit (die semantische Analyse) für jede neue Nachricht in der Eingabe-Queue durchgeführt werden muss. Da die im Repository gespeicherten XRL+-Dokumente sehr selten geändert werden, wäre dieser Ansatz in Bezug auf die Leistung des Propagationssystems sehr ungünstig.

Die zweite Alternative wäre, die semantische Analyse der XRL+-Dokumente während des Einfügens des Propagationsskripts ins Repository durchzuführen. Auf diese Weise wird dieser Vorgang nur einmal pro Propagationsskript ausgeführt. Da die Verwaltung der Propagationsskripte zu den Aufgaben des Abhängigkeitsmanagers gehört, wäre die

semantische Analyse eine Erweiterung des Konzepts des Abhängigkeits-Managers. Während der Erstellung des Propagationsskripts können mit Hilfe von Benutzerinteraktionen die Konfliktoperationen in der eingegebenen XRL+-Skripten aufgelöst und Optimierungen am Code vorgenommen werden.

Die zweite Alternative ist in Bezug auf die Leistung des Propagationssystems sicherlich die bessere.

7.1.3 Bewertung

Eine semantische Analyse bringt eine Leistungsverbesserung des Propagationssystems, wenn die XRL+-Dokumente überflüssige Operationen enthalten und diese durch die Analyse entfernt werden können. Die semantische Analyse ist für ein produktives Propagationssystem unbedingt notwendig. Ein produktives System, das zur Integration der Informationssysteme eines Unternehmens eingesetzt wird, soll in der Lage sein die angesprochenen Konflikte in den XRL+-Dokumenten und die überflüssigen Operationen innerhalb dieser Dokumente zu entfernen.

7.1.4 Erkennung überflüssiger Operationen während der Ausführung der Prozessinstanzen

Während einer semantischen Analyse können durch Analyse des Kontrollflusses des XRL+-Dokuments Transformations- und die Filterungsoperationen erkannt werden, deren Ergebnisse nicht weiter verwendet werden.

In Abbildung 7.2 werden zwei gleichen Transformationen dargestellt, die nacheinander ausgeführt werden. Die zweite Operation ist in diesem Fall überflüssig. Leider können solche überflüssige Operationen nicht während der semantischen Analyse der XRL+-Dokumente erkannt werden. Solche Operationen können immer während der Ausführung der Prozessinstanz erkannt werden. Die Prozessinstanz muss lediglich überprüfen, ob die Fälle, die in Abbildung 7.3 oder in Abbildung 7.4 dargestellt sind, nicht eingetreten sind. In Abbildung 7.3 werden die Eingabedaten für die zweite Transformation durch eine weitere Transformation (hier kann auch eine Filterungsoperation stehen) geändert. In Abbildung 7.4 werden die Ausgabedaten der ersten Transformation durch eine weitere Transformation geändert, so dass die zweite Transformation noch einmal ausgeführt werden muss.

```
…
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
…
```

Abbildung 7.2: Erkennung überflüssiger Operationen im XRL+-Dokument

```
…
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
..
<xrl:TRANSFORM xml_in= "tmp_data" xml_out= "in_data" xslt="xslt"/>
..
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
…
```

Abbildung 7.3: Erkennung überflüssiger Operationen im XRL+-Dokument (2)

```
…
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
..
<xrl:TRANSFORM xml_in= "tmp_data" xml_out="out_data" xslt="xslt"/>
..
<xrl:TRANSFORM xml_in="in_data" xml_out="out_data" xslt="xslt"/>
…
```

Abbildung 7.4: Erkennung überflüssiger Operationen im XRL+-Dokument (3)

7.2 Strategien zur Leistungssteigerung

In Abbildung 7.5 ist ein vereinfachtes Modell des Propagationssystems dargestellt. Das Propagationssystem besteht aus einer Warteschlange von Nachrichten, die noch zu bearbeiten sind und dem Propagationsmanager, der für die Bearbeitung der Nachrichten zuständig ist.

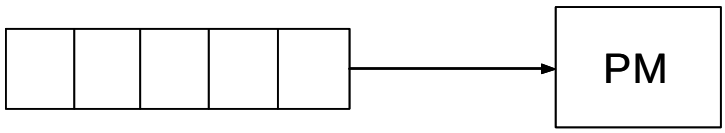

Abbildung 7.5: Leistungsmodell des Propagationssystems

Anhand des Leistungsmodells des Propagationssystems, werden im Folgenden unterschiedlichen Strategien zur Leistungssteigerung des Propagationssystems diskutiert. Es gibt drei globale Parameter, die bei diesem Modell geändert werden können, um die Leistung des Propagationssystems zu steigern: die Bearbeitungszeit einzelner Nachrichten, den Durchsatz des Systems und die Wartezeit einzelner Nachrichten in der Warteschlange des Propagationssystems.

- Durch Optimierungen des Propagationssystems oder durch Änderung der Systemkonfiguration des Propagationssystems, um die Bearbeitungszeit zu optimieren, können Nachrichten schneller bearbeitet werden. Infolge dessen steigt den Durchsatz des Systems und die durchschnittliche Wartezeit der Nachrichten in der Warteschlange des Propagationssystems wird reduziert.
  Die Optimierungen am Propagationssystem wurden im vorigen Kapitel diskutiert. Es verbleibt nur die Möglichkeit die Systemkonfiguration zur verändern.

- Die Verbesserung des Durchsatzes des Propagationssystems hat einen direkten Einfluss auf die durchschnittliche Wartezeit der Nachrichten in der Warteschlange. Diese wird kürzer. Versucht man allerdings den Durchsatz durch Erhöhung der Parallelität im Propagationssystem zu verbessern, ist mit längeren Bearbeitungszeiten der Nachrichten zu rechnen.

- Die Verbesserung der Bearbeitungszeit und des Durchsatzes des Systems führen direkt zur Verkürzung der durchschnittlichen Wartezeit der Nachrichten in der Warteschlange des System. Er gibt weiterhin Ansätze, die durch Änderung der Bearbeitungsreichenfolge der wartenden Nachrichten zu einer Verkürzung der Wartezeit einzelner Nachrichten (Nachrichten mit höherer Priorität) (siehe Abschnitt 7.3) führen.

7.3 Änderung der Bearbeitungsstrategie des Propagationsmanagers

Die durch die Informationssysteme gesendeten Nachrichten werden durch den Propagationsmanager nacheinander bearbeitet. Die Wartezeit einer Nachricht in der Eingabe-Queue ist Bestandteil der Antwortzeit dieser Nachricht im Propagationssystem. Jede neue Nachricht muss zuerst in der Eingabe-Queue eine bestimmte Zeit warten, bis sie vom Propagationsmanager bearbeitet werden kann. Je mehr zu bearbeitende Nachrichten vor dieser Nachricht in der Eingabe-Queue stehen, desto größer werden die Warte- und Antwortzeiten.

7.3.1 Der Ansatz

Für manche Geschäftsprozesse eines Unternehmens kann die Wartezeit der Nachrichten in der Warteschlange des Propagationssystems fatale Folgen für das Unternehmen haben.
Es gibt zwei Möglichkeiten die Wartezeit der Nachrichten zu reduzieren. Die erste besteht darin, den Durchsatz des Systems zu erhöhen. Strategien zur Verbesserung des Durchsatzes werden im nächsten Abschnitt diskutiert. Die zweite Möglichkeit, die Wartezeit für bestimmte Nachrichten, zu verkürzen besteht darin, diese Nachrichten zuerst zu bearbeiten, die für die Geschäftsprozesse des Unternehmens kritisch sind.

Die Idee dieses Ansatzes ist in Abbildung 7.6 dargestellt. Die Nachrichten aus der Eingabe-Queue des Propagationssystems werden je nach ihrer Wichtigkeit für die Geschäftsprozesse des Unternehmens in zwei Warteschlangen (man kann auch mehr als zwei Warteschlangen verwenden) weitergeleitet. Die Nachrichten in der Warteschlange mit der höheren Priorität werden zuerst bearbeitet. Die Nachrichten mit einer niedrigen Priorität werden erst dann bearbeitet, wenn die andere Warteschlange leer ist. Falls während der Bearbeitung von Nachrichten mit einer niedrigeren Priorität eine neue Nachricht mit einer höheren Priorität in das Propagationssystem eintrifft, kann die Bearbeitung der letzten angehalten werden oder die Priorität der entsprechenden Threads der Prozessinstanzen heruntergesetzt wird, bis die Nachricht mit der höheren Priorität bearbeitet ist. Dieser Ansatz ist in der Literatur als *priority scheduling* [CG01] bekannt.

Die Umsetzung des Konzepts kann durch die Realisierung eines *Schedulers* für das Propagationssystem erfolgen. Seine Aufgabe ist die Nachrichten nach Prioritäten zu ordnen und ihre Bearbeitung durch den Propagationsmanager zu steuern.

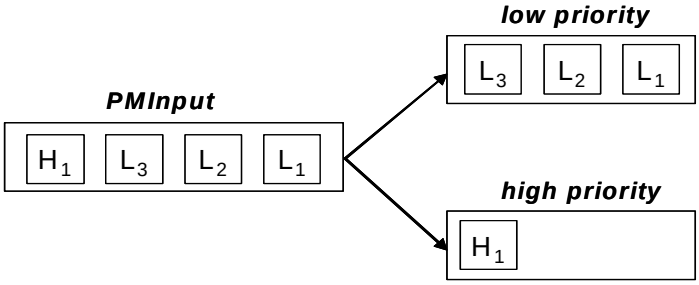

Abbildung 7.6: Prioritäten

7.3.2 Bewertung

Damit die Wartezeit bestimmter Nachrichten mit Hilfe dieses Ansatzes verkürzt wird, ist es notwendig eine Trennung der Nachrichten nach Prioritäten zu erreichen. Die Leistung des Propagationsmanagers bleibt hierbei unverändert, wenn alle Nachrichten als kritisch (mit höherer Priorität) eingestuft werden. Damit das Propagationssystem die Priorität einer Nachricht bestimmen kann, müssen zusätzliche Informationen im Repository gespeichert werden. Zum Beispiel können den Systemen oder den Schemata Prioritäten zugeordnet werden. Wenn eine Nachricht von einem System oder einem Schema generiert ist, wird ihr die Priorität des Systems oder des Schemas zugeordnet. Das führt letztendlich zu einem höheren Administrationsaufwand.

7.4 Verteilungsstrategien des Propagationssystems

Der Durchsatz des Propagationssystems lässt sich durch die Einführung zusätzlicher Kapazitäten im System erreichen.

7.4.1 Statische Aufteilung der Arbeitslast

Die Leistungsanforderungen an das Propagationssystem sind von den notwendigen Integrationsmaßnahmen der Informationssysteme eines Unternehmens abhängig und können sich von Unternehmen zu Unternehmen unterscheiden. Daher ist es zu erwarten, dass ein Propagationsmanager nicht ausreichend wäre, alle Anforderungen für die Integration innerhalb eines Unternehmens zu erfüllen. Aus diesem Grund sollte man das Propagationssystem nicht nur als eine Computer-Anwendung betrachten sondern auch als Teil der Infrastruktur des Unternehmens, dessen Aufgabe es ist, die verlässliche Zusammenarbeit vieler unabhängiger Informationssysteme zu gewährleisten.
Genauso wie bei der Planung der anderen Bestandteile der Infrastruktur, wie z.B. der lokalen Netzwerke oder der im Unternehmen verwendeten Middleware, kann man auch bei der Planung des Propagationssystems vorgehen. Äußerst wichtig ist es dabei zu wissen, welche von den möglichen Engpässen des Propagationssystems tatsächlich auftreten können.

Vielleicht die einfachste Lösung den Durchsatz des Propagationssystems zu steigern und die Ausführungszeiten einzelner Nachrichten zu reduzieren ist in Abbildung 7.7 dargestellt. Bei dieser Lösung werden die Informationssysteme in zwei oder mehrere disjunkte Mengen aufgeteilt. Jeder Menge von Systemen wird dann ein Propagationsmanager mit einer eigenen Eingabe-Queue zugeordnet. Dadurch ergibt sich eine statische Verteilung der Arbeitslast auf die verfügbaren Propagationsmanager. Um die Arbeitslast auf die Propagationsmanager zu verteilen, kann man von Mittelwerten der durch die einzelnen Systeme erzeugten Arbeitslast ausgehen.

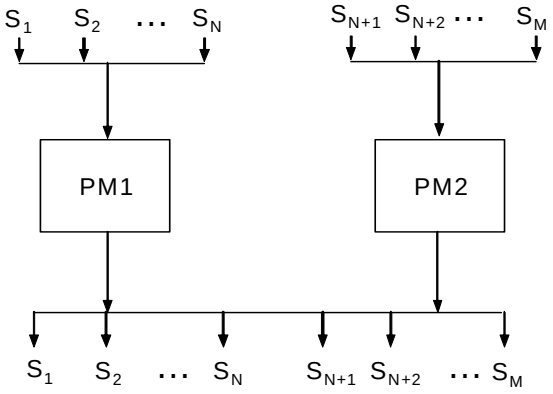

Abbildung 7.7: Aufteilung der Quellsysteme auf zwei Propagationsmanager

Im dargestellten Beispiel sind die Informationssysteme in zwei Mengen aufgeteilt und die Datenpropagationen werden durch zwei Propagationsmanager abgearbeitet. Eine statische Aufteilung der Arbeitslast ist möglich, wenn die Charakteristik der Arbeitslast (z.B. die Häufigkeit des Sendens einzelner Systemen) im Voraus bekannt ist.

7.4.2 Last Balancierung (Load Balancing)

Die statische Verteilung der Last auf mehrere Propagationsmanager hat den Nachteil, dass die Propagationsmanager möglicherweise nicht gleichmäßig ausgelastet werden. Vor allem kurzfristige Schwankungen der Arbeitslast können zu enormen Differenzen in den Auslastungen der einzelnen verfügbaren Propagationsmanager führen.

Unter Last Balancierung (engl. Load Balancing) in diesem Zusammenhang versteht man die dynamische Verteilung der Arbeitslast auf die verfügbaren Propagationsmanagers. Die Funktionsweise dieses Konzepts ist in Abbildung 7.8a) dargestellt. Die eintreffenden Nachrichten werden durch mehrere Propagationsmanager bearbeitet. Die Datenänderungenspropagation wird auf dem Propagationsmanager ausgeführt, der am wenigsten ausgelastet ist. Die ideale Situation wird erreicht, wenn alle Propagationsmanager gleichmäßig ausgelastet sind.

Die Vorteile der Last Balancierung für das Propagationssystem sind:

- höherer Durchsatz des Propagationssystems und
- kürzere Wartezeiten der Nachrichten in der Eingangswarteschlange.

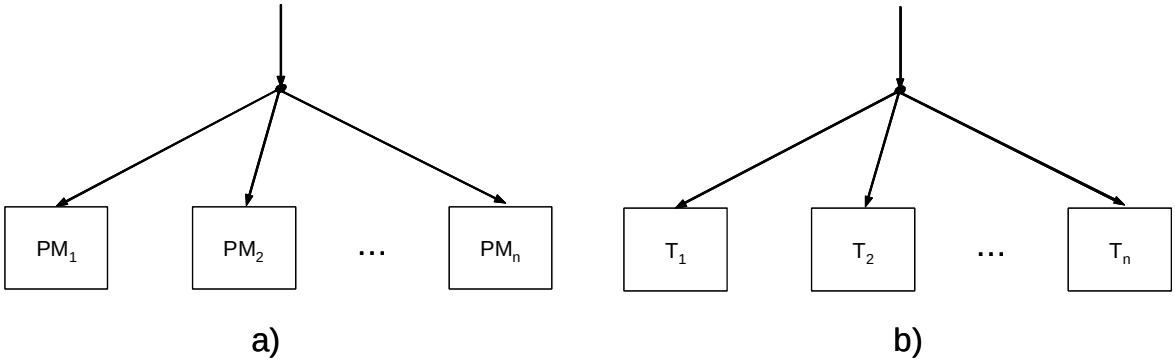

Abbildung 7.8: Load Balancing

Die Arbeit der Propagationsmanager im Propagationssystem muss koordiniert werden. Die Koordination des Propagationsmanagers besteht darin die Bearbeitung jeder neuen Nachricht auf dem Propagationsmanager auszuführen, dessen Auslastung am kleinsten ist. Nicht nur die Leistungssteigerung sondern auch die Verfügbarkeit des Propagationssystems hängt vom gewählten Koordinationsansatz [CG01] der Propagationsmanager ab. [2]

Die Koordination durch eine zentrale Koordinator-Komponente, die die Information über die Arbeit aller Propagationsmanager im System verwaltet, ist relativ einfach zu realisieren und die Kommunikationskosten zwischen dem Koordinator und den Propagationsmanager sind relativ gering. Die Verfügbarkeit des Systems mit einer zentralen Komponente hängt von der Verfügbarkeit dieser Komponente ab.
Bei einem dezentralen Koordinationsansatz der einzelnen Propagationsmanager würde die Verfügbarkeit des gesamten Propagationssystems steigen. Es ist zu erwarten, dass die Kommunikationskosten zwischen den einzelnen Propagationsmanagern wiederum steigen werden.

Last Balancierung kann man nicht nur auf der Ebene des Propagationsmanagers durchführen, sondern auch auf der Ebene des Transformers oder des Filters. Der in Abbildung 7.8 b) dargestellten Ansatz ist besonders geeignet, um die Auslastung des Transformers eines Propagationsmanagers auf weitere Transformer zu verteilen.

7.4.3 Konfiguration und Skalierung der Middleware

Die vorigen zwei Abschnitte diskutierten die Steigerung des Durchsatzes durch Verteilung des Propagationssystems. Diese Strategien führen grundsätzlich zur Leistungssteigerung des Systems, wenn die Propagationsengine, der Transformer oder der Filter der Engpass im Propagationssystem sind. Diese Strategien lösen allerdings nicht die Probleme, die durch die eingesetzte Middleware (MOM) verursacht werden. Das Problem wird anhand eines Beispiels verdeutlicht. Angenommen das Propagationssystem soll ausgelegt werden, um zehn Propagationen pro Zeiteinheit zu bearbeiten, wobei der Durchsatz der Middleware nur fünf Propagationen pro Zeiteinheit erlaubt. In diesem Fall ist die Middleware zumindest einer der Engpässe im Propagationssystem. Die anderen Engpässe im System können durch die Verteilung des

[2] Das Thema wird an dieser Stelle nicht mehr weiter vertieft, da die Verteilungsstrategien das Thema einer weiteren Forschungsarbeit sind.

Propagationssystems beseitigt werden. Um den Engpass in der Middleware zu vermeiden, ist eine Skalierung des MOM notwendig.

7.5 Zusammenfassung der Verbesserungsvorschläge

Alle Verbesserungsvorschläge, die in den vorigen Abschnitten diskutiert wurden, werden in Tabelle 7.1 zusammengefasst.

Verbesserungs-vorschlag	**Ziel**	**Leistungssteigerung**
Semantische Analyse der Prozessdefinitionen	Ausführungszeit der Prozessinstanzen senken	Ausführungszeit wird verbessert, falls die XRL+-Dokumente überflüssige Transformations- und Filterungsoperationen enthalten.
Änderung der Bearbeitungsstrategie	Wartezeit kritischer Nachrichten verkürzen	Die Wartezeit einzelner Nachrichten können verkürzt werden, falls sich den Informationsfluss zwischen den verbundenen Informationssystemen nach Prioritäten ordnen lässt.
Load Balancing des PM	Durchsatz des Propagationsmanagers verbessern	Der Durchsatz des Propagationsmanagers wird verbessert, falls die Middleware nicht der Engpass des Systems ist.
Load Balancing des Transformers	1. Durchsatz des Transformers verbessern 2. Durchsatz des Propagationssystems verbessern	Ziele 1 und 2 werden erfüllt, falls der Transformer der Engpass im System ist.
Konfiguration und Skalierung die Middleware	Durchsatz des Propagationsmanagers verbessern	Der Durchsatz des Propagationsmanagers wird verbessert, falls die Middleware der Engpass im System ist.

Tabelle 7.1: Bewertung der Verbesserungsvorschläge

Kapitel 8

Zusammenfassung

Die Aufgabenstellung zu dieser Diplomarbeit erfolgte im Rahmen des Teilprojekts A5 des Sonderforschungsbereichs 467 „Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion“. Das Aufgabengebiet des Teilprojekts ist die informationstechnische Unterstützung wandlungsfähiger Unternehmen. Zu diesem Zweck wurde ein Prototyp, das “Stuttgarter Informations- und Explorationssystem” (SIES), entwickelt.

Das Ziel dieser Arbeit war die Leistungserfassung und Leistungssteigerung des Propagationsmanagers. Der Propagationsmanager ist Bestandteil des Stuttgarter Information- und Exploration Systems (SIES) und steuert den gezielten Austausch geänderter Daten zwischen unterschiedlichen Informationssystemen. Um das gesetzte Ziel zu erreichen, wurde die Aufgabestellung in mehrere Teilaufgaben gegliedert. Im Folgenden wird ein Überblick über die Ergebnisse dieser Arbeit gegeben.

8.1 Ergebnisse der Arbeit

Als Erstes wurde ein Konzept zur Leistungserfassung und Leistungsanalyse des Propagationsmanagers und seiner Komponenten erstellt. Auf Basis dieses Konzepts wurde eine Testumgebung realisiert, mit deren Hilfe Messungen am Propagationssystem ermöglicht wurden. Zur Untersuchung der Leistung des Propagationsmanagers mussten neben der Testumgebung auch unterschiedliche Testfälle erstellt werden. Die Testfälle wurden entwickelt, um das System zu analysieren und seine Schwachstellen zu finden. Insbesondere wurden bei den Messungen die Bestandteile der Bearbeitungszeit und die Auslastung der unterschiedlichen Komponenten des Propagationsmanagers bei Änderungspropagationen untersucht. Außerdem dienten die erstellten Testfälle als Vergleichsbasis für die am Propagationsmanager umgesetzten Verbesserungen.

Ausgehend von den gemessenen Daten wurden anschließend Optimierungsmaßnahmen getroffen, um die Leistung des Propagationssystems zu steigern. Zunächst wurde die Arbeit der so genannten Prozess-Konnektoren optimiert, was zu einer deutlichen Reduzierung der Initialisierungszeiten der Prozessinstanzen geführt hat. Weiterhin wurden zwei Caches eingeführt. Das Caching der Prozessdefinitionen hat die Kosten, die beim Parsen der XRL+-Dokumente entstehen, verringert. Durch die Einführung des Stylesheet-Cache wurde die Arbeit des Transformers optimiert. Das Caching hat einen weiteren Vorteil für die Leistung des Propagationsmanagers - die Reduzierung der Anzahl der Repository-Zugriffe.

8.2 Ausblick

Die Leistungssteigerung des Propagationsmanagers lässt sich neben Optimierungen auch durch andere Strategien erreichen. In Kapitel 7 wurden weitere Konzepte zur Leistungssteigerung des Propagationsmanagers diskutiert, die aus zeitlichen Gründen nicht im Rahmen dieser Arbeit umgesetzt wurden. Diese Konzepte können folgendermaßen zusammengefasst werden:

- Mit Hilfe einer semantischen Analyse der XRL+-Dokumente können überflüssige Operationen in der Prozessbeschreibung entdeckt und eliminiert werden.

- Die zweite diskutierte Strategie zur Leistungssteigerung besteht darin die Bearbeitungsreihenfolge der in die Eingabe-Queue eintreffenden Nachrichten zu ändern. Die Bearbeitung sollte dabei nach Prioritäten geordnet werden. Die Nachrichten, die für die Geschäftsprozesse eines Unternehmens kritisch sind, sollen zuerst bearbeitet werden.

- Das dritte Konzept zur Leistungssteigerung sollte durch die Einführung zusätzlicher Kapazitäten im Propagationssystem umgesetzt werden.

Anhang A

Im Folgenden werden die XML-Schemata der Systeme und die Transformationsskripten aufgelistet, die für die Testfälle im Kapitel 5 verwendet werden.

Schema 1

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Kunden">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Kunde" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Umsatz" type="xs:decimal"/>
                     <xs:element name="Adress">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Land" type="xs:string"/>
                              <xs:element name="Stadt" type="xs:string"/>
                              <xs:element name="PLZ" type="xs:string"/>
                              <xs:element name="Strasse" type="xs:string"/>
                              <xs:element name="Nummer" type="xs:string"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Schema 2

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Customers">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Customer" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Name" type="xs:string"/>
```

```
                    <xs:element name="Konto" type="xs:decimal"/>
                    <xs:element name="Address">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="State" type="xs:string"/>
                          <xs:element name="city" type="xs:string"/>
                          <xs:element name="PLZ" type="xs:string"/>
                          <xs:element name="Street" type="xs:string"/>
                          <xs:element name="Number" type="xs:string"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
</xs:schema>
```

Schema 3

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="TotalUmsatz">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Umsatz" type="xs:decimal"/>
        <xs:element name="Monat" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Transformationsskript 1

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
        <xsl:template match="Kunden">
                <TotalUmsatz>
                        <xsl:apply-templates/>
                </TotalUmsatz>
        </xsl:template>

        <xsl:template match="Kunde">
                <xsl:element name="Umsatz">
                        <xsl:value-of select="sum(Umsatz)"/>
                </xsl:element>
                <xsl:element name="Monat">
                        <xsl:value-of select="//@monat" />
                </xsl:element>
```

```
        </xsl:template>
</xsl:stylesheet>
```

Transformationsskript 2

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
        <xsl:template match="Kunden">
                <Customers>
                        <xsl:apply-templates/>
                </Customers>
        </xsl:template>

        <xsl:template match="Kunde">
                <xsl:element name="Customer">
                        <xsl:element name="Name">
                                <xsl:value-of select="Name"/>
                        </xsl:element>
                        <xsl:element name="Konto">
                                <xsl:value-of select="Umsatz"/>
                        </xsl:element>
                        <xsl:element name="Address">
                                <xsl:element name="State">
                                        <xsl:value-of select="Adress/Land"/>
                                </xsl:element>
                                <xsl:element name="city">
                                        <xsl:value-of select="Adress/Stadt"/>
                                </xsl:element>
                                <xsl:element name="Street">
                                        <xsl:value-of select="Adress/Strasse"/>
                                </xsl:element>
                                <xsl:element name="Number">
                                        <xsl:value-of select="Adress/Nummer"/>
                                </xsl:element>
                        </xsl:element>
                </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```

Transformationsskript 3

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

        <xsl:template match="Kunden">
                <Kunden>
                <xsl:apply-templates select="Kunde">
                        <xsl:sort order="ascending" select="Umsatz" data-
type="decimal">
```

```
                </xsl:sort>
            </xsl:apply-templates>
            </Kunden>
        </xsl:template>

        <xsl:template match="Kunde">
            <xsl:element name="Name">
                <xsl:value-of select="Name"/>
            </xsl:element>
            <xsl:element name="Umsatz">
                <xsl:value-of select="Umsatz"/>
            </xsl:element>
            <xsl:element name="Adress">
                <xsl:element name="Land">
                    <xsl:value-of select="Adress/Land"/>
                </xsl:element>
                <xsl:element name="Stadt">
                    <xsl:value-of select="Adress/Stadt"/>
                </xsl:element>
                <xsl:element name="Strasse">
                    <xsl:value-of select="Adress/Strasse"/>
                </xsl:element>
                <xsl:element name="Nummer">
                    <xsl:value-of select="Adress/Nummer"/>
                </xsl:element>
            </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```

Anhang B

Das Datenbankschema des MessageDictionary

Im Folgenden werden die SQL-Statements zum Erzeugen der vier Relationen des Datenbankschemas des MessageDictionary aufgeführt.

```
CREATE TABLE "SYSTEMMESSAGES"  (
        "ID" BIGINT PRIMARY KEY,
        "NAME" VARCHAR(100) NOT NULL ,
        "SYSTEM" VARCHAR(100) ,
        "SCHEMA" VARCHAR(100) NOT NULL ,
        "TYPE" VARCHAR(10) ,
        "MESSAGES" SMALLINT WITH DEFAULT 1 ,
        "MESSAGE_ID" BIGINT NOT NULL ) ;


ALTER TABLE "SYSTEMMESSAGES"
        ADD CONSTRAINT "UNIQUE_NAME" UNIQUE ("NAME");


CREATE TABLE "MESSAGES"  (
        "ID" BIGINT PRIMARY KEY,
        "CONTENT" CLOB(2097152) NOT LOGGED NOT COMPACT );


CREATE TABLE "MESSAGEGROUPS"  (
        "ID" BIGINT PRIMARY KEY,
        "MSGID" BIGINT NOT NULL ,
        "GROUPID" BIGINT NOT NULL );


CREATE TABLE "GROUPS"  (
        "ID" BIGINT PRIMARY KEY,
        "NAME" VARCHAR(100) NOT NULL );


ALTER TABLE "MESSAGEGROUPS"
        ADD CONSTRAINT "MSGGRP_GRP" FOREIGN KEY ("GROUPID")
        REFERENCES "GROUPS"("ID")
        ON DELETE CASCADE
        ON UPDATE NO ACTION;


ALTER TABLE "MESSAGEGROUPS"
        ADD CONSTRAINT "MSGGRP_MSG" FOREIGN KEY ("MSGID")
        REFERENCES "MESSAGES" ("ID")
```

```
ON DELETE CASCADE
ON UPDATE NO ACTION;
```

Literaturverzeichnis

[AW00] W. van der Aalst, A. Kumar, XML Based Schema Definition for Support of Interorganisational Workflow, In *Proceedings of 21st International Conference on Application and Theory of Perty Nets (ICATPN 2000)*, Aarhus, Denmark, June 2000.

[BA90] A.Bobbio, System Modelling with Petri Nets, *Systems Reliability Assessment. Proceedings of the Ispra Course, 1988, Madrid, Spain*, pages 103-143. Dordrecht, Netherlands: Kluwer, 1990.

[BB86] B.Bunday, Basic Queueing Theory, Edward Arnold,1986.

[CC00] C. Constantinescu, U. Heinkel, H. Meinecke, A Data Change Propagation System for Enterprise Application Integration, 2000.

[CC01] C. Constantinescu, U. Heinkel, R. Rantzau, B. Mitschang, SIES: An Approach for a Federated Information System in Manufacturing, *In Proceedings of the International Symposium on Information Systems and Engineering (ISE)*, 2001.

[CG01] G. Coulouris, J.Dollimore, T.Kindberg, Distributed Systems: Concepts and Design, Addison-Wesley, 2001.

[DP91] P. Denning, Queueing in Networks of Computers, American Scientist, May-September,1991.

[FD78]D. Ferrari, Computer Systems Performance Evaluation, Prentice-Hall,Inc, New Jersey, 1978.

[HP02] P. Horváth, Controlling, 8.Auflage, München 2002.

[HS02] S. Heldt, Funktionale Erweiterungen und Entwicklung einer Graphischen Benutzer-oberfläche eines Propagationsmanagers, Diplomarbeit Nr. 1979, Universität Stuttgart, Fakultät Informatik, 2002.

[HT98] T. Härder, Rahm, E. (1998). Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer, 1998.

[HU00] U. Heinkel, Informationsmodelle für wandlungsfähige Informationssysteme, Diplomarbeit Nr. 1846, Universität Stuttgart, Fakultät Informatik, 2000.

[GM02] M. Gillmann, R. Mindermann, G. Weikum, Benchmark and Configuration of Workflow Management Systems, CoopIS, 186-197, 2000.

[KG99] G.Kotsis, Performance Management in Parallel and Distributed Computing Systems, 1999.

[KO01] O. Kersten, Konzeption eines Propagationsmanagers, Diplomarbeit Nr. 1937, Universität Stuttgart, Fakultät Informatik, 2001.

[LH92] H. Langendörfer, *Leistungsanalyse von Rechensystemen: Messen, Modellieren, Simulation*, München; Wien: Hanser, 1992.

[LK02] K.Lagally, Grundlagen der Betriebsysteme, Vorlesungsskript, Universität Stuttgart, Fakultät Informatik, 2002.

[MB03] B. Mitschang, E.Westkämper: Devide et Impera: A Flexible Integration of Layout Planning and Logistics Simulation through Data Change Propagation. In *The 36th CIRP International Seminar on Manufacturing systems*, Saarbruecken, Germany, June 2003.

[MH02] H. Meinecke, Entwurf und Implementierung eines Abhängigkeits-Managers, Diplomarbeit Nr.1971, Universität Stuttgart, Fakultät Informatik, 2002.

[MS02] S. Minnert, XML&Co. Die W3C-Spezifikation für Dokumenten- und Datenarchitektur, Addison-Wesley, 2002.

[SFB03] Homepage des Sonderforschungsbereiches 467, http://www.sfb467.unistuttgart.de, Universität Stuttgart, 2003.

[SUN2] SUN Microsystems: Java™ API for XML Processing (JAXP), http://java.sun.com/xml/jaxp.html.

[SUN01] Sun Microsystems Inc. Java Message Service, Version 1.1.2b http://java.sun.com/products/jms, August 2001

[TPC] Transaction Processing Performance Councel, http://www.tpc.org.

[TA92] A. Tanenbaum, Modern Operating Systems, Prentice-Hall, 1992.

[W3C99] World Wide Web Consortium (W3C),XML Path Language (XPath) Version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath, November 1999.

[W3C00] World Wide Web Consortium (W3C), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/REC-xml, Oktober 2000.

[XSL03] http://javaboutique.internet.com/resources/books/JavaXSLT/, 2003.

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, den 20. April 2004

(Gueorgui Kostov Ovtcharov)