Universität Stuttgart
Software–Labor
Projekt 1.1:
Workflow–Management–Systeme
Breitwiesenstraße 20–22
D-70565 Stuttgart

Fakultätsbericht Nr. 1996/19
Software–Labor Bericht Nr. SL–5/96
CR–Klassifikation H.2.0, H.2.4

Beschreibung des prototypisch implementierten Workflowsystems Surro[1]

Stefan Schreyjak
Stefan.Schreyjak@informatik.uni-stuttgart.de
Hubert Bildstein

20. Dezember 1996

[1]Diese Arbeit wird von der IBM Deutschland Entwicklung GmbH und dem Ministerium für Wissenschaft und Forschung, Baden Württemberg, unterstützt.

Zusammenfassung

In diesem Bericht wird das im Rahmen des Software–Labors, Projekt Workflow–Management, erstellte Workflowsystem Surro vorgestellt. Zielsetzung des Projekts ist es, Konzepte zur Verbesserung der Fehlertoleranz von Workflowsystemen zu evaluieren. Dazu wurde auf Basis des FlowMark–Workflowmodells eine Workflow–Engine, ein Aktivitätenmanager, ein Workflow–Session–Manager und ein Workflow–Monitor erstellt.

In der Workflow Spezifikation können Sphären (Gruppen von Aktivitäten) definiert werden, die zur Erhöhung der Fehlertoleranz dienen. Es gibt zwei Arten von Sphären. In den Transaktions–Sphären sind alle Operationen der Engine und alle Tätigkeiten in den Anwendungsprogrammen, die in den Aktivitäten verwendet werden, in einer großen Workflow–Transaktion geschützt. Wenn während der Bearbeitung der Sphäre ein Fehler auftritt, wird die Sphäre automatisch in ihren Initialzustand zurückgesetzt und alle Datenänderungen werden rückgängig gemacht. In den Kompensations–Sphären müssen alle Aktivitäten eine zusätzliche Kompensationsaktivität besitzen, die die Auswirkungen der normalen Aktivität kompensieren können. Wenn ein Fehler in dieser Sphäre auftritt, dann werden alle notwendigen Kompensationsaktivitäten automatisch durch das Workflowsystem aktiviert.

Nach einer Einführung in die Aufgabenstellung und Terminologie von Workflow–Management werden die Workflow–Transaktionen und Kompensations–Sphären als Konzepte zur Fehlertoleranz ausführlich vorgestellt. Ihre Implementierung und die dabei gesammelten Erfahrungen im Umgang mit der CORBA–Implementierung von IBM (DSOM 3.0) und dem Transaktionsdienst (OTS) werden beschrieben. Das zugrundeliegende Datenmodell für die Beschreibung der Workflows wird erläutert. Der Aufbau des Workflowsystems Surro wird aufgezeigt und einzelne Implementierungsaspekte werden beschrieben. Das System wurde mit einem ebenfalls beschriebenen Beispielworkflow zur Bearbeitung einer Beschwerde bei einer Kreditkartenabrechnung getestet.

Inhaltsverzeichnis

1. Workflow–Management–Systeme

In diesem Abschnitt werden die wesentlichen Begriffe auf dem Gebiet des Workflow–Managements eingeführt [WfM96]. Anschließend wird motiviert, welche Vorteile der Einsatz von Workflowsystemen bringt. Aufgrund eines Vergleichs mit der heutzutage existierenden betrieblichen Softwareausstattung wird die Kernidee des Workflow–Managements herausgestellt.

1.1. Begriffe

Ein **Workflow–Management–System** (WFMS) ist ein Softwaresystem zur Koordination und kooperativen Abwicklung von Geschäftsvorgängen in verteilten heterogenen Rechnerumgebungen. Die Aufgaben eines Workflow–Management–Systems liegen in einer ersten Phase in der Modellierung der Aufbau– und Ablauforganisation eines Unternehmens und in der zweiten Phase in der Steuerung, Überwachung und Protokollierung der modellierten Abläufe.

Die Ablauforganisation wird formal in **Geschäftsprozessen** modelliert, in denen die Reihenfolgebeziehungen der einzelnen Vorgangsschritte spezifiziert werden. Zu jedem Schritt wird bestimmt, welche Arbeitsobjekte (Daten bzw. Dokumente) und welche menschlichen und technischen Ressourcen zur Ausführung benötigt werden. Ein Geschäftsprozeß bzw. ein **workflow** kann als Graph modelliert werden mit Vorgangsschritten als Knoten und Kontrollfluß– und Datenflußbeziehungen als Kanten.

Eine Vorgangsschritt, im Geschäftsprozeß **Aktivität** genannt, ist ein Stück zusammenhängender Arbeit, die von einer Person ausgeführt wird. Zur Bearbeitung der Aufgabe in einer Aktivität können vom Bearbeiter interaktive Anwendungsprogramme eingesetzt werden. Alternativ sind auch manuelle Aktivitäten ohne Computerunterstützung möglich, oder automatische Aktivitäten, die ohne menschliche Interaktion auskommen.

1.1.1. Modell eines Geschäftsprozesses

In Abbildung 1.1 ist das Modell eines Geschäftsprozesses dargestellt. Hierbei wird das von FlowMark benutzte Modell vorgestellt [LR94]. Die Aktivitäten sind durch Kontrollflußkonnektoren miteinander verbunden. Die Aktivität A_2 wird sequentiell nach Beendigung von A_1 ausgeführt. A_3 und B_1 werden parallel ausgeführt. Nach A_1 wird

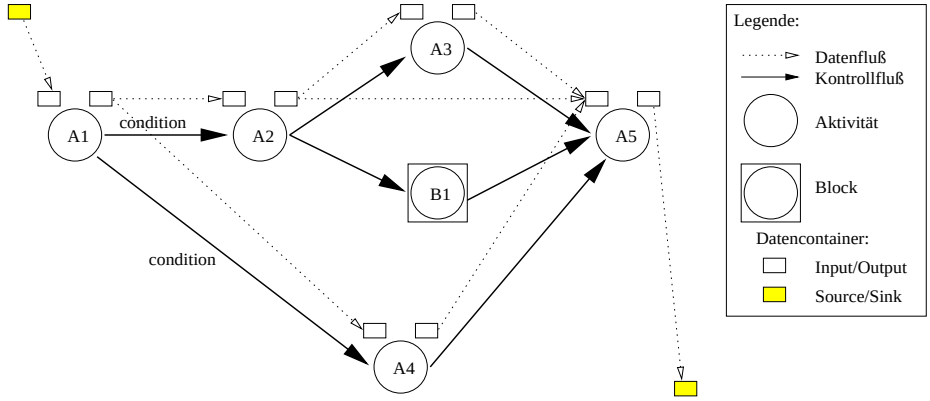

Abbildung 1.1.: Das Modell eines Geschäftsprozesses

entsprechend den Bedingungen an den Kontrollkonnektoren der obere Zweig und/oder der untere Zweig mit A_4 ausgeführt. Die Bedingungen sind frei wählbar. Es sind also entweder beide (parallele Ausführung), eine von beiden (alternative Ausführung) oder keine (Ende der Ausführung) der Bedingungen erfüllt. Nach Ausführung eines Zweiges wird A_5 bearbeitet, wobei vor Ausführung von A_5 alle dessen eingehenden Konnektoren evaluiert sein müssen. Jede Aktivität besitzt Inputcontainer und Outputcontainer für Daten. Die Weitergabe von Daten wird durch die Datenflußkonnektoren gesteuert. Die speziellen Datencontainer Source und Sink sind die Input– und Outputcontainer eines Geschäftsprozesses.

1.1.2. Modell einer Aktivität

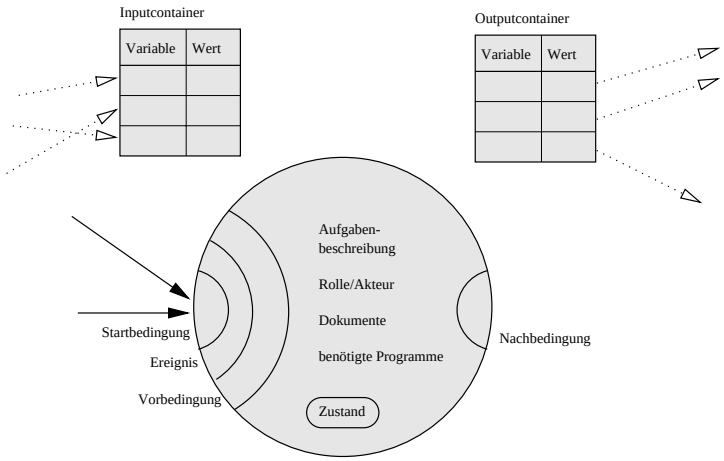

Abbildung 1.2.: Die innere Struktur einer Aktivität

In Abbildung 1.2 ist eine Aktivität mit ihrer inneren Struktur dargestellt. Der Dateninputcontainer besteht aus einer Menge von Variablen, die zu Beginn einer Aktivität

über die Datenflußkonnektoren belegt werden, und von der Anwendung ausgelesen werden können. Die Variablen im Datenoutputcontainer werden durch das Anwendungsprogramm gesetzt. Die eingehenden Kontrollflußkonnektoren werden über die Startbedingung logisch miteinander verknüpft. Falls die Startbedingung wahr wird, wird auf das Eintreten eines Ereignisses gewartet. Falls dieses eingetreten ist oder falls kein Ereignis angegeben ist, wird die Vorbedingung geprüft. Falls auch diese erfüllt ist, wird die Aktivität einem Bearbeiter zugeteilt und diesem auf die Arbeitsliste gelegt. Der oder diejenige führt das zugehörige Programm aus, das dann den Datencontainer als die Eingabedaten verarbeitet und die Ausgabedaten in den Outputcontainer schreibt. Anhand der Nachbedingung kann das System noch Kontrollen durchführen, ob die gestellte Aufgabe wirklich erfolgreich erledigt worden ist. Falls nicht, erhält derselbe Bearbeiter die Aktivität erneut.

1.1.3. Die Systemkomponenten eines Workflowsystems

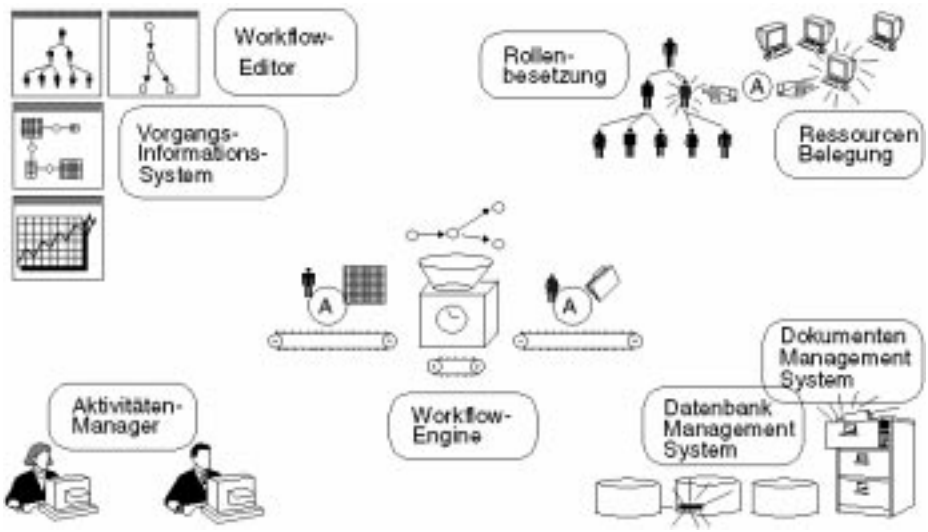

Abbildung 1.3.: Die Systemkomponenten eines Workflowsystems

Die zentrale Komponente eines Workflowsystem ist die Workflow–Engine, die die Spezifikation eines Geschäftsprozesses einliest, diesen instanziiert und die Aktivitäten zu den richtigen Zeitpunkten auf die Bearbeiter verteilt, sowie für den Transport der Daten und Dokumente zu den Bearbeitern sorgt. Die Anwendungsdaten und Dokumente, sowie die Workflow–Verwaltungsdaten werden in Datenbank– und Dokumenten–Management–Systemen gespeichert. Die Engine benutzt eine Organisationskomponente, die Kenntnis über die Aufbauorganisation des Unternehmens hat, zur Auflösung einer Rolle (z.B. Sachbearbeiter) in einen konkreten Bearbeiter. Zudem kann die Organisationskomponente noch eine Ressourcenverwaltung realisieren. Die Spezifikation der Geschäftsprozesse und die Aufbauorganisation werden im

Workflow–Editor grafisch eingegeben. Das Vorgangsinformationssystem liefert verschiedene Sichten auf den aktuellen Zustand des Workflowsystems und auf die Historie. Der Aktivitätenmanager stellt die Benutzeroberfläche des Bearbeiters zum System dar. Durch ihn kann der Bearbeiter seine ihm zugeteilten Aktivitäten ausführen und neue Vorgänge starten.

1.2. Motivation von Workflowsystemen

Durch die Identifikation und die anschließende Spezifikation von Geschäftsprozessen werden nicht nur lokale, sondern auch globale Optimierungspotentiale im Unternehmen aufgedeckt, die durch die Neuordnung ganzer Wertschöpfungsketten ausgenutzt werden können. Mit der Abkehr von abteilungsorientierten hin zu prozeßorientierten Organisationsstrukturen wird die Automatisierung von Prozessen erleichtert. Die unternehmensweite Steuerung und die verbesserten Informationsfähigkeiten des Systems erlauben es, den Zustand aller Vorgänge detailliert zu kontrollieren. Die Automatisierung der Prozesse birgt das Potential zu einem erheblich geringeren Anteil von Transport- und Liegezeiten im Gesamtvorgang. Die informationstechnische Modellierung der Prozesse erlaubt eine wesentlich flexiblere, schnellere und einfachere dynamische Anpassung an geänderte Randbedingungen. Die Integration der für ein Unternehmen lebenswichtigen bestehenden Computeranwendungen (legacy systems) ist möglich.

Der Einsatz von Workflowsystem kann so zu einer hohen Produktivitätssteigerung führen.

1.3. Die Kernidee des Workflow–Management

Heutzutage läßt sich die Situation der betrieblichen EDV in vielen Fällen so charakterisieren:

Die Mitarbeiter einer Firma werden funktionsorientiert durch Computerprogramme in ihrer Arbeit unterstützt. Typische Beispiele sind hier Programme für die Unterstützung der Büroarbeit, wie Tabellenkalkulation oder Textverarbeitung. Die kooperative Bearbeitung eines Dokuments wird durch solche Programme im allgemeinen nicht angeboten. Die Vorgeschichte der Bearbeitung und die noch zu leistenden zukünftigen Arbeiten finden in funktionsorientierten Systemen keine Unterstützung. Die Computerprogramme werden als „Insellösungen“ eingesetzt: Es besteht kaum ein Bezug zum gesamten Vorgang.

Neben diesen funktionsorientierten Programmen existieren noch stärker prozeßorientierte Anwendungen, die meist als große monolithische Anwendungen auf Hostrechnern realisiert sind. Solche Anwendungen implementieren „hartverdrahtet“ einen oder mehrere Geschäftsvorgänge. Andere Geschäftsprozesse können damit nicht ausgeführt werden und Änderungen im Geschäftsprozeß können nur mit großem Aufwand

eingebracht werden.

Die Idee der Workflowsysteme ist nun, den Prozeßbegriff explizit im Softwaresystem sichtbar zu machen und auf diese Weise ein konfigurierbares System für die Abwicklung von Geschäftsprozessen zu schaffen. Der Geschäftsprozeß wird dazu in Arbeitsschritte aufgegliedert, in denen funktionsorientiert gearbeitet wird. Die Arbeitsschritte werden durch Kontroll– und Datenfluß–Beziehungen zu einem Prozeß verknüpft. Das Workflowsystem ermöglicht die Modellierung und flexible Änderung dieser Beziehungen und stellt bei der Ausführung des Prozesses deren Einhaltung sicher. Zusätzlich nutzt das System noch weitere Informationen, wie z. B. die Aufbauorganisation, um eine bestmögliche Prozeßunterstützung und Kontrolle zu gewährleisten.

Die Kernidee der Datenbanksysteme ist das Herauslösen des Datenmanagements aus den Anwendungen. Analog dazu lösen Workflowsystem das Prozeßmanagement aus den Anwendungen.

Das hat den Vorteil, daß das Workflowsystem bei der Ausführung beliebiger Geschäftsprozesse verwendet werden kann, und daß Änderungen aufgrund des Ansatzes „Modellierung statt Programmierung“ leichter zu realisieren sind.

2. Fehlertoleranz in Workflowsystemen

Die Einführung eines Workflow–Management–Systems in ein Unternehmen muß wohlüberlegt und sorgfältig geplant sein. Die wertschöpfenden Prozesse innerhalb des Unternehmens werden dadurch unter die Kontrolle des Workflow–Management–Systems gestellt. Der Erfolg des Unternehmens hängt somit direkt von der Funktionsfähigkeit des Systems ab. Wenn das System einmal nicht funktionsfähig sein sollte, kommen alle computerunterstützten Geschäftsprozesse zum Erliegen. Der mögliche Ausweg, die Prozesse kurzfristig ohne Computerunterstützung durchzuführen, ist meist nicht einfach gangbar, da es dadurch zu Inkonsistenzen zwischen den Daten im System und der Realität kommt. Nach dem Neustart des Systems kann es im allgemeinen nicht sofort wieder eingesetzt werden, da zuerst der veraltete Datenzustand manuell auf den neuesten Stand gebracht werden muß. Durch eine schrittweise Einführung und durch den Einsatz eines fehlertoleranten und stabilen Systems kann man dieser Gefahr begegnen. In [SB96] wird die hier angesprochene Problematik ausführlich behandelt.

2.0.1. Allgemeine Anforderungen

Ein Workflow–Management–System muß als das „Rückgrat“ eines Unternehmens angesehen werden: Ein Bruch wäre tödlich.

Oberste Strategie beim Einsatz eines solchen Systems muß daher die Fehlervermeidung sein. Da in der realen Welt dieses Ziel aber nicht vollständig erreicht werden kann, benötigt man darüber hinaus Mechanismen, um auf Fehler reagieren zu können. Das ganze System muß daher folgende allgemeine Anforderungen erfüllen.

- **Korrektheit:**
  Ein System ist korrekt, wenn es die Aufgabe, für die es spezifiziert ist, erfüllt. Voraussetzung dafür ist unter anderem, daß die Integrität der Daten, die durch das System verwaltet werden, gewährleistet ist. Nur mit konsistenten Daten kann ein System korrekt arbeiten. Inkonsistente Daten können zu fehlerhaftem Verhalten führen.

  Auch beim Auftreten von Fehlern darf das System keine inkonsistenten

Zustände erzeugen. Diese Forderung hat große Auswirkungen auf die Fehlerbehandlungsmechanismen des Workflowsystems.

Diese Anforderung kann nicht alleine vom Workflowsystemen erfüllt werden, da es keine vollständige Kontrolle über die bearbeiteten Daten hat. In den Aktivitäten können Programme oder Menschen Daten außerhalb der Kontrolle des Workflowsystem inkonsistent verändern, ohne daß das System darauf Einfluß hat oder die Inkonsistenz überhaupt bemerkt.

- **Hohe Zuverlässigkeit:**
  Ein System ist hoch zuverlässig, wenn es über lange Zeiträume hinweg ohne Auftreten eines effektiven Fehlers funktioniert. Ein Fehler ist dann effektiv, wenn er die spezifizierte Funktion des Systems beeinträchtigt [GR93].

- **Hohe Verfügbarkeit:**
  Längerfristige Ausfallzeiten des Gesamtsystems verringern die Verfügbarkeit. Während dieser Zeit kann keiner der Mitarbeiter eines Unternehmens oder einer Behörde weiterarbeiten, da bei dem umfassenden Einsatz eines Workflow–Management–Systems nahezu alle Arbeiten über das System oder zumindest mit dessen Hilfe abgewickelt werden. Der Stillstand des Systems kann daher zu immensen Kosten führen. Die Verfügbarkeit kann durch hohe Zuverlässigkeit oder durch den Einsatz redundanter Komponenten erhöht werden. Durch den Einsatz von Mechanismen für einen schnellen, weitgehend automatischen Wiederanlauf des Systems (Recovery) kann auch die Verfügbarkeit erhöht werden.

- **Robustheit:**
  Ein robustes System verhält sich tolerant gegenüber unerwarteten Eingaben und bleibt auch in Ausnahmesituationen weiterhin funktionsfähig. Die Eigenschaft Robustheit trägt damit zur Erhöhung der Verfügbarkeit bei.

- **Hohe Flexibilität:**
  Ein flexibles System erlaubt auch nach dem Auftreten eines Fehlers oder einer Ausnahmesituation, die nicht automatisch durch Fehlerbehandlungsmechanismen des Systems beseitigt werden können, manuelle Eingriffe in die Kontrolle des Systems, um den Fortgang der Prozesse zu ermöglichen.

  In einem Workflowsystem werden immer auch Fehler auftreten, die sich nicht automatisch beheben lassen. Das sind zum einen Systemfehler, wie der Ausfall eines Rechnerknotens, und zum anderen Fehler, die das System nicht erkennen kann, da sie semantischer Natur sind. In einem solchen Fall ist der Benutzer auf die Flexibilität des Systems angewiesen: Der Eingriff eines Menschen ist nötig. Er muß die Kontrolle übernehmen und das System in einen Zustand überführen, in dem es die Kontrolle wieder selbst übernehmen kann. Dazu muß das System Methoden anbieten, die der Benutzer „manuell“ anwenden kann, um die Fehlersituation zu bereinigen und den Prozeß wieder in geordnete

Bahnen zu lenken. Kompetente Benutzer können so unter Ausnützung ihres Fachwissens und dem Einsatz von Software–Werkzeugen die Auswirkungen von Fehlern beseitigen und so das System reparieren.

Aufgrund der Bedeutung des Problemgebiets der Fehlertoleranz bei Workflowsystemen wurden die Konzepte *Workflow–Transaktionen* und *Kompensations–Sphären* entwickelt und im Workflowsystem Surro implementiert. In den folgenden Abschnitten werden diese zwei Konzepte erläutert.

2.1. Workflow–Transaktionen

Probleme bei der Verwendung von ACID–Transaktionen in langandauerenden Vorgängen sind der hohe Verlust bereits geleisteter Arbeit bei einem „rollback“ und die Akkumulation von Sperren auf Daten, was zu schlechterer Kooperation führt [SB96]. Der Einsatz von Transaktionen eignet sich also nicht als alleiniges Konzept zur Implementierung von fehlertoleranten Workflows. Wenn man diese Probleme berücksichtigt, zeigt sich aber, daß dieses Konzept für einen begrenzten Einsatz in Workflowsystemen durchaus geeignet sein kann. Das Einsatzgebiet ergibt sich aus den folgenden Randbedingungen:

- Die Transaktionen müssen “klein” sein. Dies bezieht sich zum einen auf eine kurze Zeitdauer, zum anderen auf die Datenmengen, die sie anfassen und somit sperren. Ansonsten greifen die bereits oben beschriebenen Probleme bei langandauernden Transaktionen.

- Alle Aktivitäten der Workflow–Transaktionen müssen als Resource–Manager an der Transaktion teilnehmen können und entsprechende Schnittstellen zur Steuerung des Recovery anbieten. Die Schnittstellen müssen zu dem im Workflowsystem verwendeten Transaktions–Service passen. Es können damit keine beliebigen Aktivitäten an einer Workflow–Transaktion teilnehmen!

2.1.1. Begriffe

Eine **Sphäre** ist eine Menge von Aktivitäten in einem Workflow. Wenn zwischen Aktivitäten Abhängigkeiten in der Art existieren, daß nie eine der Aktivitäten erfolgreich und eine andere erfolglos beendet werden darf, können die Aktivitäten zu einer Sphäre zusammengefaßt werden. Eine Sphäre wird zur Modellierungszeit des Workflows spezifiziert. In der Abbildung 2.1 ist eine Sphäre in einem Aktivitätennetz eingezeichnet.

Eine Sphäre muß dabei keine Zusammenhangskomponente im Aktivitätennetz bilden. Es müssen also nicht alle Aktivitäten in einer Kontrollflußbeziehung stehen, wie in Abbildung 2.2 gezeigt.

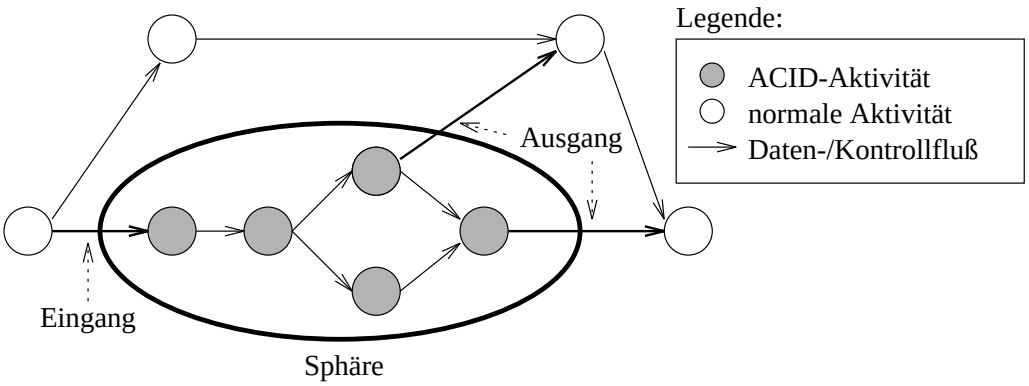

Abbildung 2.1.: Eine Sphäre in einem Aktivitätennetz

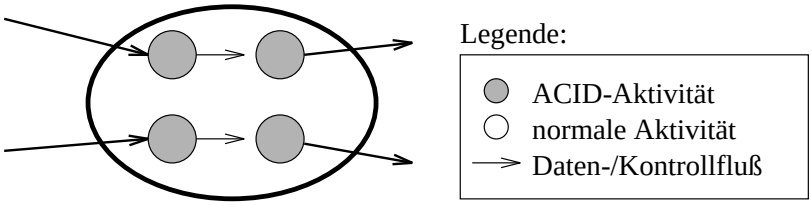

Abbildung 2.2.: Eine Menge von Aktivitäten in einer Sphäre, die keine Zusammenhangskomponente im Aktivitätennetz bilden

Als **Eingang** einer Sphäre wird der Kontroll– bzw. Datenfluß bezeichnet, der von einer Aktivität außerhalb der Sphäre zu einer Aktivität innerhalb der Sphäre führt. Entsprechend wird als **Ausgang** der Kontroll– bzw. Datenfluß definiert, der von einer Aktivität innerhalb der Sphäre zu einer Aktivität außerhalb der Sphäre führt. Eine Sphäre kann mehrere Ein– und Ausgänge besitzen. In der Abbildung 2.1 hat die Sphäre einen Eingang und zwei Ausgänge.

Als **ACID-Aktivitäten** werden Aktivitäten bezeichnet, die die Eigenschaft haben, daß sie entweder selbst Resource–Manager sind oder nur auf Daten über Resource–Manager zugreifen. Es muß gewährleistet sein, daß die in den Aktivitäten verwendeten Daten nur über Resource–Manager im Rahmen einer Transaktionen gelesen oder verändert werden (siehe Abschnitt 2.1.3).

Definition: *Workflow–Transaktion*

> *Eine Workflow–Transaktion ist eine Menge von Aktivitäten (eine Sphäre im Workflowmodell), die im Kontext einer ACID–Transaktion ausgeführt werden.*

Die Bearbeitung einer Aktivität kann durch das folgende grundlegende Zustandsübergangsdiagramm modelliert werden (siehe Abbildung 2.3). Das Diagramm ist zum besseren Verständnis gegenüber realen Implementierungen vereinfacht worden. Ein Startzustand ist am linken Rand schwarz markiert. Ein Endzustand ist rechten Rand markiert.

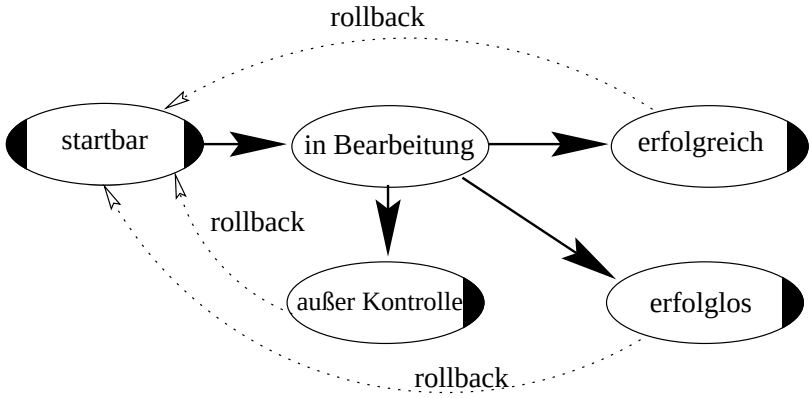

Abbildung 2.3.: Das (vereinfachte) Zustandsübergangsdiagramm einer Aktivität

Eine Aktivität beginnt mit dem Initialzustand STARTBAR. Der Start durch den Bearbeiter überführt die Aktivität in den Zustand IN BEARBEITUNG. Dort können Fehler auftreten, die auf der Aktivitätenebene angesiedelt sind. Wenn ein solcher Fehler erkannt wird, wird die Aktivität in den Zustand ERFOLGLOS gebracht. Auf diese Art bekommt das Workflowsystem Kenntnis vom Auftreten eines Aktivitätenfehlers und ein Fehlerkontext kann von der Aktivität an das Workflowsystem übergeben werden. Kann eine Aktivität ohne Auftreten eines Fehlers beendet werden, kommt sie in den Zustand ERFOLGREICH. Eine Aktivität wird in den zusätzlichen Zustand AUSSER KONTROLLE gebracht, wenn die Aktivität weder eine erfolgreiche, noch eine erfolglose Bearbeitung melden kann. In diesem Fall ist die Aktivität außer Kontrolle geraten. Problematisch ist hierbei die Detektion des Übergangs IN BEARBEITUNG nach AUSSER KONTROLLE. In diesem Fall muß das Workflowsystem ohne Hilfe der Aktivität erkennen, daß ein Aktivitätenfehler aufgetreten ist. Voraussetzung für ein fehlertolerantes Workflowsystem ist, daß das System auch dann sinnvoll weiterarbeiten kann, wenn einmal dieser Zustand auftritt. Innerhalb von Transaktions–Sphären gibt es noch weitere Zustandsübergänge, die implizit durch ein Rollback der Sphären–Transaktionen ausgelöst werden.

2.1.2. Das Konzept der Workflow–Transaktion

Ein Workflowsystem, das Workflow–Transaktionen anbietet, tritt als Starter und als normaler Teilnehmer der Transaktion auf. Beim Betreten einer Sphäre muß die Workflow–Transaktion durch das Workflowsystem bei einem Transaktions–Service initiiert werden. Dann registriert sich das Workflowsystem selbst als Teilnehmer. Dazu muß das Workflowsystem als Resource–Manager für die Workflow–Daten auftreten können.

Falls das Workflowsystem Datenflüsse verwaltet, über die Aktivitäten mit Daten versorgt werden, muß das System dafür sorgen, daß die Daten die Sphäre nicht vorzeitig verlassen. Dasselbe gilt für den Kontrollfluß. Erst mit dem erfolgreichen Ende

der Transaktion dürfen Aktivitäten außerhalb der Sphäre angestoßen werden. Das Workflowsystem muß als Resource–Manager die Isolationseigenschaft der Transaktion bereitstellen, indem die Daten– und Kontrollflüsse der Sphäre nach außen bis zum erfolgreichen Ende verzögert werden.

Durch die Teilnahme an der Transaktion kann das Workflowsystem den Bearbeitern eine Funktion an ihrer Bedienoberfläche anbieten, mit der sie eine Workflow–Transaktion interaktiv abbrechen können.

Falls die Workflow–Transaktion zurückgesetzt werden soll, muß das Workflowsystem den Zustand des Workflows wieder in den Anfangszustand der Sphäre bringen. Alle bisherigen Änderungen innerhalb der Sphäre müssen durch das Workflowsystem rückgängig gemacht werden. Nach dem Abbruch und Rücksetzen der Sphäre wird die Workflow–Transaktion durch das Workflowsystem neu gestartet. Anstatt eines Neustarts sind auch andere Aktionen denkbar. So könnte man z. B. den Neustart x–mal versuchen und nach dem x–ten Fehlschlag eine alternativen Workflow starten.

Eine Workflow–Transaktion muß zusammen mit dem Workflow in der Modellierungskomponente des Workflowsystems spezifiziert werden. Eine Workflow–Transaktionen wird durch eine Sphäre modelliert. Alle Aktivitäten, die an Transaktion teilnehmen sollen, müssen in eine Sphäre aufgenommen werden. Die Modellierungskomponente muß auch dafür sorgen, daß folgende strukturellen Bedingungen für die Sphäre eingehalten werden:

- Alle Aktivitäten der Sphäre sind Teilnehmer an der Workflow–Transaktion. Die Modellierungskomponente muß daher prüfen, ob die an der Sphäre teilnehmenden Aktivitäten die entsprechenden Randbedingungen erfüllen, wie sie in Abschnitt 2.1.3 beschrieben werden.

- Eine Schachtelung von Sphären ist erlaubt und dient zur Verkleinerung des Bereichs, der zurückgesetzt werden soll. So kann ein feineres Recovery–Granulat unterstützt werden. Da einmal spezifizierte Workflows in Form von Subprozessen wiederverwendbar sein sollen und in diesen Prozessen auch Sphären definiert sein können, benötigt man auch aus diesem Grund die Möglichkeit geschachtelter Workflow–Transaktionen.

  Innerhalb der Aktivitäten können durch Anwendungsprogramme neue Transaktionen begonnen und wieder beendet werden. Diese Transaktionen sind dann als in die Workflow–Transaktion geschachtelte Transaktionen zu realisieren.

- Aus der Isolationseigenschaft der Sphäre ergibt sich, daß kein Pfad von einem Ausgang auf einen Eingang derselben Sphäre existieren darf (Abb. 2.4a).

  Angenommen, es gäbe einen solchen Pfad. Aufgrund der Isolation wird der Kontrollflußausgang erst nach Beendigung der Sphäre aktiv. Die Sphäre kann aber noch nicht beendet sein, da der Kontrollflußeingang auf diesem Pfad noch nicht aktiv sein kann, d. h. es gibt eine nicht beendete Aktivität in der Sphäre.

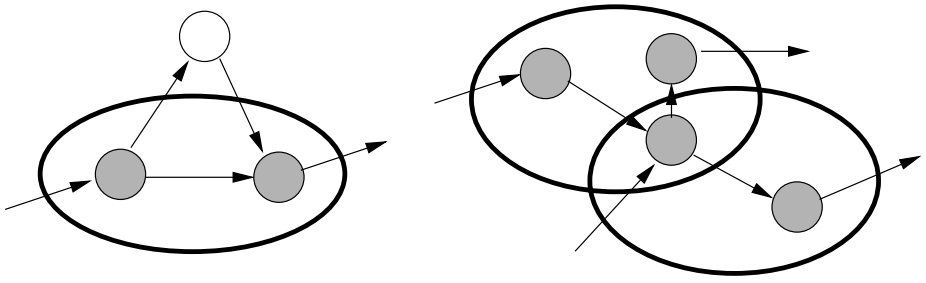

Abbildung 2.4.: Verlassen und Wiedereintritt des Kontrollflusses (a) und partiell überlappende Sphären (b) sind nicht erlaubt.

Die Sphäre kann noch nicht beendet sein. Es gibt einen Widerspruch, daher darf kein solcher Pfad existieren.

- Die partielle Überlappung von Sphären (Abb. 2.4b) ist nicht möglich. Eine Aktivität darf immer nur an höchstens einer Sphäre teilnehmen. Partiell überlappende Sphären können durch eine Vereinigungsoperation in eine einzige Sphäre überführt werden. Partiell überlappende Sphären erweisen sich somit als unnötig. Der Effekt des feineren Transaktionsgranulats kann durch geschachtelte Sphären ebenso erreicht werden.

2.1.3. Anforderungen an ACID–Aktivitäten

Damit Aktivitäten an einer Workflow–Transaktion teilnehmen können, müssen sie bestimmten Voraussetzungen genügen. Wir unterscheiden deshalb zwischen normalen Aktivitäten und sogenannten *ACID–Aktivitäten*, die diese Voraussetzungen erfüllen.

- Die Aktivitäten dürfen nur über Resource–Manager auf Daten zugreifen. Wenn sie Daten selbst verwalten, müssen die Aktivitäten selbst als Resource–Manager auftreten. Eine Aktivität, die als Resource–Manager agieren möchte, muß alle notwendigen Funktionen implementiert haben, um selbständig ein Recovery ausführen zu können.

- Die Aktivitäten müssen eine geeignete Schnittstelle aufweisen, über die bestimmte Funktionen der Aktivitäten ausgelöst werden können (z. B. das Recovery). Ebenso müssen sie Schnittstellen für die Teilnahme an einem 2–Phasen–Commit–Protokoll besitzen (Prepare, Commit, Rollback). Damit erreicht der Transaktions–Service eine gemeinsame Übereinkunft aller an einer Transaktion beteiligten Resource–Manager über den Erfolg oder Mißerfolg der Transaktion. Die angebotene Schnittstelle muß zu dem im Workflowsystem verwendeten Transaktions–Service passen.

- Wenn in den Aktivitäten physische Operationen aufgeführt werden, bedarf es der Verwendung eines erweiterten Resource–Manager, der *Physical–Resource–Manager* (PRM) genannt wird [Sch93]. "Real actions" haben im Gegensatz zu Datenbankoperationen die Eigenschaft, daß ihre Auswirkungen sofort sichtbar werden und daß diese Auswirkungen oft nicht mehr rücksetzbar sind. Das klassische Beispiel für eine solche Operation ist das Bohren eines Loches in ein Werkstück. Unter der Annahme, daß nur eine einzige physische Operation in der Workflow–Transaktion stattfindet, kann das Recovery eines PRM so aussehen: Wenn der Abbruch der Transaktion vor der physischen Operation stattfindet, dann muß der PRM wie ein regulärer Resource–Manager reagieren. Es wird ein Rollback durchgeführt. Wenn der Abbruch nach der Ausführung der physischen Operation stattfindet, wird wiederum ein normales Rollback durchgeführt. Die physische Operation wird dabei nicht zurückgesetzt. Beim wiederholten Starten der Transaktion wird dann die bereits in der vorherigen Transaktion ausgeführte physische Operation ausgelassen. Wenn der Abbruch während der physischen Operation stattfindet, dann muß eine anwendungsspezifische Fehlerbehebungsmaßnahme durch den PRM getroffen werden.

  Wenn man mehrere physische Operation innerhalb einer Workflow–Transaktion benutzen will, wird die Komplexität des Recovery deutlich höher.

  Für den Fall, daß die Auswirkungen physischer Operation verzögert werden können, z.B. das Verschicken einer Email oder eines Briefes, ist es Aufgabe des Resource–Managers, dafür zu sorgen, daß die Operation erst in der Propagierungs–Phase des Zwei–Phasen–Commit–Protokolls am Ende der Workflow–Transaktion ausgeführt wird. So wird die Email solange verzögert, bis die gesamte Transaktion erfolgreich beendet wird. Die Operation darf dann allerdings nicht mehr fehlschlagen.

2.1.4. Einsatzgebiete von Workflow–Transaktionen

Durch den alleinigen Einsatz von Workflow–Transaktionen kann man das Ziel eines fehlertoleranten Ablaufs von Geschäftsprozessen nicht erreichen. Die Nachteile und Einschränkungen, wie schon zu Beginn von Abschnitt 2.1 beschrieben, überwiegen in diesem Fall meist die Vorteile bis hin zur Unbenutzbarkeit. Nur in einem eng beschränkten Einsatzfeld erweist sich das Konzept der Workflow–Transaktionen als hilfreich. In Abschnitt A.1 wird ein Beispielprozeß näher beschrieben, in dem zwei Workflow–Transaktionen zum Einsatz kommen. Das Umfeld des Einsatzes liegt im Datenbankbereich, wo im allgemeinen schon die entsprechenden Resource–Manager mit den standardisierten Schnittstellen vorhanden sind, und in dem auch die Anwendungen häufig Sicherheitsanforderungen haben, die nur durch den Einsatz von Transaktionen abgedeckt werden können.

Wichtig erweist sich dieses Konzept auch bei der Verwendung von sogenannten *business objects* als Aktivitäten. Business–Objekte sind Repräsentanten für alle in

einem Geschäftsprozeß vorkommenden Objekte. Dies können Programme, Personen oder Daten in der traditionellen Sichtweise sein. Business–Objekte werden zur Zeit in der BOMSIG special interest group der OMG (Object Management Group) standardisiert. Diese Objekte bieten Methodenaufrufe an, um Operationen auf Daten durchzuführen. Die Methoden sind oftmals von kurzer Dauer und werden automatisch ausgeführt, d.h. es gibt kaum manuelle Interaktion. Diese Methodenaufrufe können in einer Workflow–Transaktion als Operationen eingebunden werden, falls die Objekte als Resource–Manager implementiert sind.

2.1.5. Einbindung von Legacy–Software

Eine Möglichkeit, vorhandene Softwareprogramme (Legacy–Software) in Workflow–Transaktionen einzubinden, stellt der Ansatz in [Täu96] dar. Der Hauptansatzpunkt besteht darin, das Dateisystem selbst zu einem Resource–Manager zu machen. Damit können alle Programme, die nur das Dateisystem zur Speicherung von Daten verwenden, in Transaktionen eingebettet werden, indem alle Dateizugriffe unter Transaktionsschutz gestellt werden. Dieses Konzept funktioniert aber nicht bei Programmen, die eine Datenbank zur Speicherung verwenden. Aber in diesen Fällen kann die Datenbank in die Workflow–Transaktion eingebunden werden.

Änderungen auf Dateien bzw. Teilen von Dateien können durch das modifizierte Dateisystem nach außen hin isoliert und bei einem Rollback wieder ungeschehen gemacht werden. Unter der Voraussetzung, daß die Anwendung ihren Zustand vollständig in Dateien sichert, ist dies eine Möglichkeit, nichttransaktional implementierte Anwendungen in Workflow–Transaktionen zu verwenden.

Die Einbindung eines solchen Konzepts in ein Workflow–Management–System erfordert, daß das Workflowsystem vor dem Start einer Alt–Anwendung dem Dateisystem als Resource–Manager mitteilt, unter welchem Transaktionskontext die Dateioperationen der Anwendung ablaufen müssen. Die Legacy–Software kann dazu keine Hilfestellung geben, da sie keine Kenntnis über Transaktionen hat. Siehe dazu Abschnitt 3.4.

2.1.6. Realisierungsansätze

Als Realisierungsansätze für Workflow–Transaktionen bieten sich die Standards für verteilte Transaktionen an: Es kommt die X/Open Spezifikation for Distributed Transaction Processing (DTP) XA und der Object Transaction Service (OTS) [OTS94] der Object Management Group (OMG) in Frage. Insbesondere im Verbund mit dem Einsatz der Business–Objekte kann sich OTS als sinnvoll erweisen. Die Aktivitäten müssen die in den Standards spezifizierten Funktionen als API anbieten.

Bei Einsatz von OTS müssen die Anwendungsprogramme in den Aktivitäten ein Objektinterface besitzen. Das Workflowsystem muß sich der Dienste eines CORBA–kompatiblen Objekt Request Brokers (ORB) bedienen, um die Anwendungsprogramme in den Aktivitäten aufzurufen.

2.2. Kompensations–Sphären

Das Konzept der Workflow–Transaktionen stellt hohe Anforderungen an die Funktionalität der Aktivitäten, die an einer Workflow–Transaktionen teilnehmen. In vielen Fällen wird aber ein Workflowsystem mit Aktivitäten eingesetzt, die nicht diesen Anforderungen entsprechen. Oftmals haben die Anwendungsprogramme, die in den Aktivitäten aufgerufen werden, kein Wissen darüber, daß sie im Rahmen eines Geschäftsprozesses eingesetzt werden. Sie können daher auch nicht oder nur sehr schwer auf die Bedürfnisse der Workflow–Transaktionen abgestimmt werden.

Aus diesem Grund ist ein weiteres Konzept bei der Bearbeitung von Workflows nötig, das den Ablauf von Workflows fehlertoleranter macht, ohne diese hohen Anforderungen zu besitzen. Der Ansatz der *Kompensations–Sphären* [Ley95] stellt kaum noch Anforderungen an die Aktivitäten. Im Gegenzug dazu muß man aber auf die Isolationseigenschaft und die garantierte Konsistenz der Anwenderdaten bei der Ausführung eines Workflow verzichten. Die Eigenschaft der Atomizität und der Dauerhaftigkeit bleiben erhalten. Das Mittel zur Erreichen dieses Ziels sind Kompensationsaktivitäten.

2.2.1. Begriffe

Eine **Kompensationsaktivität** unterscheidet sich nur durch ihre Verwendung von einer normalen Aktivität. Jede Kompensationsaktivität muß einer normalen Aktivität bzw. einer Sphäre zugeordnet sein und soll alle Auswirkungen der normalen Aktivität bzw. der gesamten Sphäre beseitigen. Das Workflowsystem bietet außer dem Aufruf der Kompensationsaktivität keine weitere Unterstützung, um dieses Ziel zu erreichen. Wegen der fehlenden Isolationseigenschaft für die Anwendungsdaten der Kompensations–Sphären muß die Kompensationsaktivität auch berücksichtigen, daß die Datenänderungen der normalen Aktivität eventuell schon von anderen Aktivitäten gelesen und zur Weiterverarbeitung benutzt worden sind. Die Kompensationsaktivität muß auch in diesen Fällen geeignete Maßnahmen treffen.

Der Begriff **Sphäre** [1] bezeichnet auch hier eine nicht unbedingt zusammenhängende Menge von Aktivitäten in einem Workflow.

Definition: *Kompensations–Sphäre* (engl.: compensation sphere)

> *Eine Kompensations–Sphäre ist eine Menge von Aktivitäten, die entweder alle im Zustand 'erfolgreich' oder alle im Zustand 'kompensiert' sind, wenn der Kontrollfluß die Sphäre verlässt.*

Das Zustandsdiagramm für die Aktivitäten muß daher wie in Abbildung 2.5 modifiziert werden. Es wird zusätzlich ein Zustand KOMPENSIERT eingeführt. Dieser Zustand ist weitgehend äquivalent zum Zustand STARTBAR mit dem Unterschied,

[1] siehe Definition Seite 17

daß mindestens eine Bearbeitung und eine Kompensation der Aktivität stattgefunden hat. Der Endzustand ERFOLGREICH wird dann verlassen, wenn andere Aktivitäten der Sphäre kompensiert werden müssen. Eine Sphäre ist dann kompensiert, wenn alle Aktivitäten der Sphäre kompensiert sind.

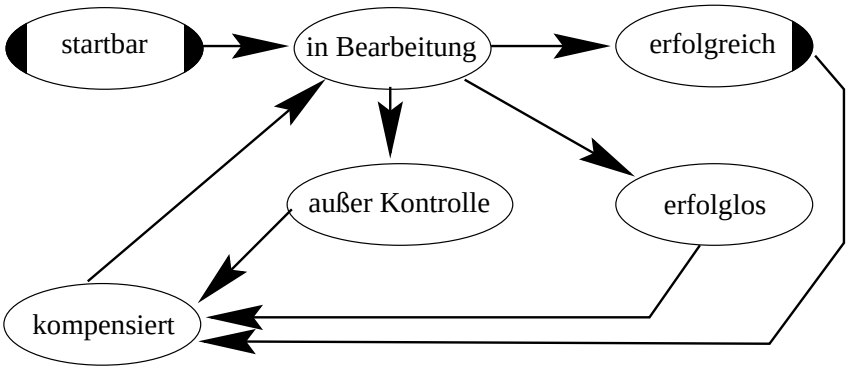

Abbildung 2.5.: Das (vereinfachte) Zustandsdiagramm für Aktivitäten bei Kompensations–Sphären

2.2.2. Das Konzept der Kompensations–Sphären

Zur Modellierungszeit werden Aktivitäten zu einer Sphäre zusammengefaßt. Aus der Sicht der Aktivitäten außerhalb der Sphäre werden die Aktivitäten zu einer atomaren Ausführungseinheit. Zum Ausführungszeitpunkt des Workflows sorgt das Workflowsystem dafür, daß die Sphäre von den Aktivitäten außerhalb der Sphäre isoliert wird, indem der Kontroll– und Datenfluß bis zum Ende der Sphäre verzögert wird. Aktivitäten außerhalb der Sphäre können somit keine Zwischenergebnisse von Aktivitäten innerhalb der Sphäre über das Workflowsystem bekommen. Da die Anwenderprogramme aber weiterhin auf beliebigen Datenbeständen arbeiten können, die nicht im Kontrollbereich des Workflowsystem liegen, können Zwischenergebnisse durchaus von anderen Programmen verarbeitet werden. Die Isolation kann daher nicht durch das Workflowsystem garantiert werden.

Wenn ein Fehler auftritt, werden alle bereits beendeten Aktivitäten kompensiert und alle Aktivitäten abgebrochen, die noch in Bearbeitung sind. Danach kann die Sphäre entweder neu gestartet werden oder es wird ein alternativer Weg im Workflow eingeschlagen, wie im vorherigen Kapitel beschrieben.

Dieses Konzept fordert schwächere Voraussetzungen an die teilnehmenden Aktivitäten als die Workflow–Transaktionen. Die Aktivitäten müssen kompensierbar sein, d. h. zu jeder Aktivität A in der Sphäre muß eine Kompensationsaktivität A^{-1} existieren, die die Auswirkungen der Aktivität A rückgängig macht. Wie diese Kompensierbarkeit erreicht wird, liegt ganz in der Verantwortung des Erstellers der Aktivität. Das Konzept der Kompensations–Sphären bietet dazu außer dem Aufruf der Kompensationsaktivität keine weitere Unterstützung an. Ergänzend kann die Anforderung

aufgestellt werden, daß jede Aktivität (bzw. das Anwenderprogramm innerhalb der Aktivität) an ihrer Schnittstelle eine Funktion anbieten muß, mit der die Aktivität vorzeitig abgebrochen werden kann, ohne daß dadurch der Anwenderdatenbestand in einem inkonsistenten Zustand hinterlassen wird. Mit dieser Anforderung kann eine Optimierung angewendet werden, die mit Hilfe des vorzeitigen Abbruchs der Aktivität die Rücksetzzeit der Sphäre verkürzt, indem unnötige Arbeit nach Auftreten eines Fehlers in der Sphäre verhindert wird. Die Aktivität A muß nach ihrem Ende oder nach einem Abbruch erneut gestartet werden können, ohne daß dadurch ein Fehler auftritt.

Die Anforderungen an die Kompensationsaktivität sind dafür aber um so ausgeprägter. Neben der Existenz die Aktivität muß auch gefordert werden, daß die Aktivität niemals fehlschlägt. Die Aufgabe der Kompensationsaktivität, das Beseitigen der Auswirkungen der Aktivität A, wird dabei in keiner Weise durch das Workflowsystem unterstützt. Der Kompensationsaktivität muß diese Aufgabe völlig selbständig und korrekt durchführen.

Eine Sphäre wird dann zurückgesetzt, wenn eine der Aktivitäten in einen Fehlerzustand (siehe Abbbildung 2.5) überführt wird. Das Zurücksetzen einer Sphäre sollte auch über den interaktiven Aufruf einer Funktion möglich sein, die dem Benutzer des Workflowsystem an der Bedienoberfläche angeboten wird. Über diese Funktion kann manuell ein Zurücksetzen ausgelöst werden, das eventuell durch spezielle Rechte abgesichert werden kann.

Nach der Auslösung des Rücksetzvorgangs ist eine Entscheidung möglich, ob die Sphäre bis zu ihrem Beginn oder zu einem weniger weit zurückliegenden Punkt zwischen den Aktivitäten zurückgesetzt werden soll. Wenn diese Auswahl, wohin zurückgesetzt werden soll, dem Benutzer interaktiv überlassen wird, dann hat man eine Art "Undo"–Funktion im Workflow realisiert. Man kann die letzten Vorgangsschritte (Aktivitäten) innerhalb der Grenzen der Sphäre rückgängig machen und dann an dem gewünschten Punkt weiterarbeiten.

Für die Ausführungsreihenfolge der Kompensationsaktivitäten gibt es verschiedene Möglichkeiten. Die Kompensationsaktivitäten können alle parallel ausgeführt werden, da alle dazu notwendigen Daten schon während der Ausführung der normalen Aktivitäten gespeichert werden. Falls ein solches Verhalten nicht gewünscht ist, können die Kompensationsaktivitäten in der umgekehrter Reihenfolge wie die normalen Aktivitäten ausgeführt werden. Diese umgekehrte Reihenfolge kann durch Umdrehen der Kontrollflußbeziehung bestimmt werden oder durch Auswerten der Startzeitpunkte aller normalen Aktivitäten. Eine dritte denkbare Methode besteht in einer frei spezifizierbaren Reihenfolge, die während der Modellierung des Prozesses festgelegt werden muß.

Die Schachtelung von Sphären muß, wie bei den Workflow–Transaktionen auch, aufgrund der Wiederverwendung von Workflowteilen erlaubt sein. Daraus ergibt sich die Notwendigkeit, auch ganze Sphären rückgängig machen zu müssen. Zusätzlich zu dem Kompensieren aller Einzelaktivitäten einer Sphäre kann man auch die Möglichkeit schaffen, mit einer einzigen Kompensationsaktivität eine ganze Sphäre auf einmal

zu kompensieren. Dazu müssen dann zu Sphären auch Kompensationsaktivitäten einführt werden.

Ein Überlappen von Sphären bedeutet, daß eine Aktivität an mehr als an einer einzigen Kompensations–Sphäre teilnimmt. Wenn man dies zuläßt, handelt man sich das Problem der kaskadierenden Kompensation weiterer Sphären ein. Da Überlappung keinen weiteren Vorteil als eine feinere Abstufung der Sphären bringt und dieser Vorteil auch über die Schachtelung von Sphären erlangt werden kann, kann das Überlappen ohne Verlust an Funktionalität verboten werden.

2.2.3. Vergleich zwischen Transaktions– und Kompensations–Sphären

Das Konzept der Kompensations–Sphären unterscheidet sich in einigen wesentlichen Punkten von dem Konzept der Workflow–Transaktionen. Kompensations–Sphären stellen hauptsächlich die Eigenschaft der Atomizität bereit. Dabei werden keine besonderen Anforderungen an die Aktivitäten gestellt. Das Workflowsystem führt keine undurchdringbare Isolation der Sphären durch. Zwischenergebnisse aus den Aktivitäten in der Sphäre können von allen Aktivitäten auf Kosten der Konsistenz genutzt werden. Es wird keine standardisierte Schnittstelle zur Einbindung in einen transaktionalen Kontext gefordert. Die Aktivitäten müssen nicht an einem Zwei–Phasen–Commit–Protokoll teilnehmen können. Die Aktivitäten müssen keine Resource–Manager sein und ihre Änderungen auf den Daten rückgängig machen können. Sie müssen kein Recovery implementiert haben.

Ein Schwachpunkt der Kompensations–Sphären ist die Tatsache, daß für eine Garantie der Rücksetzbarkeit einer Sphäre gefordert werden muß, daß Kompensationsaktivitäten nicht fehlschlagen dürfen. Diese Forderung ist aber nur schwer verwirklichbar. Die Dauerhaftigkeit der Ergebnisse kann zwischen der Bearbeitung von Aktivitäten gesichert werden, da dann das Workflowsystem die Kontrolle hat. Wenn das System aber während der Bearbeitung einer Aktivität abstürzt, gehen alle Änderungen verloren, die in den laufenden Aktivitäten gemacht wurden. Auf diese Weise kann ein inkonsistenter Datenzustand entstehen, der eventuell zur Folge hat, daß die anschließende Wiederholung der Aktivität fehlschlagen kann.

Kompensations–Sphären können entweder über die vorhandenen Mittel der Workflow–Spezifikationssprache modelliert oder durch eine direkte Unterstützung in der Workflow–Engine realisiert werden. Beim Modellierungsansatz werden die Kompensationsaktivitäten wie normale Aktivitäten behandelt. Der Kontrollfluß zwischen den Kompensationsaktivitäten muß explizit festgelegt werden. Im zweiten Ansatz wird der Zustand einer Sphäre durch die Engine verwaltet. Die Engine stößt bei Bedarf die Kompensationsaktivitäten an. Es ist keine weitere Spezifikationen des Kontrollflusses nötig, es sei denn, man möchte die Reihenfolge der Kompensationsaktivitäten explizit festlegen.

Der Vorteil der Kompensations–Sphären liegt darin, daß bei einer Realisierung

nur Änderungen im Workflowsystem nötig sind. Auf die Realisierung der Aktivitäten hat dieses Konzept keine Auswirkungen.

3. Motivation weiterer Workflow–Konzepte

Eine Überprüfung des Konzepts der Workflow–Transaktionen und der Kompensations–Sphären waren die Hauptbeweggründe für die Erstellung des Surro Prototyps. Dadurch ergab sich die Gelegenheit, auch noch weitere Konzepte zu verwirklichen und auf ihre Tauglichkeit zu testen. Die weitergehenden Konzepte, die in Surro implementiert wurden, werden in diesem Kapitel beschrieben.

3.1. Ereignisse

3.1.1. Problemstellung und Lösungsansätze

Ereignisse (Events) sind ein wichtiges Konzept in Workflowsystemen, das die Reaktion auf Umwelteinflüsse, die Synchronisation zwischen Workflows und Aktivitäten, sowie die Ausnahmebehandlung erlaubt.

Durch das Einführen von Ereignissen kann ein Workflowsystem auf Einflüsse der Umwelt, d. h. der Welt außerhalb des Workflowsystems, reagieren. Ein typisches Beispiel für einen solchen externen Einfluß auf ein Workflowsystem ist das Eintreffen eines Briefes mit der Post. Dieser Ereignis muß vom System erkannt werden. Eine entsprechende Reaktion muß darauf stattfinden können. Die Aufgabe der Ereignisverarbeitungskomponente eines Workflowsystems besteht im Erkennen von Ereignissen, im Verteilen der Ereignisse und im Reagieren auf Ereignisse. Typische Reaktionen auf das Auftreten eines Ereignisse sind der Start einer Aktivität oder eines ganzen Geschäftsprozesses. Ereignisse dienen somit als Auslöser (Trigger) einer Aktion.

Ereignisse können auch zur Synchronisation zwischen unterschiedlichen Geschäftsprozessen eingesetzt werden. Es kann vorkommen, daß ein Prozeß solange angehalten werden muß, bis sich ein anderer Prozeß in einem bestimmten Zustand befindet. Zusätzlich zur reinen Synchronisation sollte an diesem Zeitpunkt auch Kommunikation (Datenaustausch) zwischen den Geschäftsprozessen stattfinden können.

Ereignisse können auch für eine Art generische Fehler- und Ausnahmebehandlung eingesetzt werden. So kann ein Fehler oder eine Ausnahme ein Signal (Ereignis) auslösen. Wenn für dieses Ereignis kein Aktion innerhalb des Geschäftsprozesses definiert ist, dann wird das Ereignis an den nächsthöheren Prozeß weitergeben. Dieser

Prozeß versucht dann, auf das Ereignis zu reagieren. Der oberste Prozeß muß für jedes definierte Signal ein Defaultaktion besitzen. Dieses Konzept ist analog zu dem Konzept der *exceptions* in den Programmiersprachen Ada oder C++.

3.1.2. Die unterschiedlichen Arten von Ereignissen

Ein Ereignis wird als ein *atomares Auftreten* [CM94] definiert. Vor dem Auftreten ist es noch nicht da, nach dem Auftreten ist es eingetreten. Zustandsänderungen der Außenwelt (aus der Sicht des Workflowsystem) werden als *externe Ereignisse* bezeichnet. Externe Ereignisse werden mit einem eindeutigen Namen identifiziert. Das Eintreten eines bestimmten Zeitpunktes wird *Zeitereignis* genannt. Zeitereignisse können absolut oder relativ sein. Ereignisse können entweder einfach oder zusammengesetzt sein. *Primitive Ereignisse* sind nicht weiter zerlegbar. *Zusammengesetzte Ereignisse* sind Verknüpfungen von einfachen oder zusammengesetzten Ereignissen. Typische Verknüpfungen können z. B. logische Operatoren sein.

Im Surro–Workflowsystem wird zwischen workflow–lokalen und workflow–globalen Ereignissen unterscheiden. Lokale Ereignisse sind nur innerhalb eines Vorgangs gültig. Globale Ereignisse haben einen globalen Gültigkeitsbereich, d. h. ein Ereignis kann in mehreren Vorgängen referenziert werden. Lokale Ereignisse können zur vorgangsinternen Synchronisation eingesetzt werden. Globale Ereignisse werden zur Synchronisation zwischen Aktivitäten unterschiedlicher Vorgänge verwendet.

3.1.3. Externe Ereignisse

Wenn ein Ereignis auftritt, meldet der Initiator das Ereignis der Workflow–Engine, die den Zeitpunkt des Auftretens des Ereignisses in die Datenbank schreibt und das Auftreten des Ereignisses an alle Knoten weitergibt, die auf dieses Ereignis warten. Ein Initiator eines Ereignisses kann z. B. ein Bearbeiter sein, der das Auftreten des Ereignisses bemerkt hat. Im Beispiel mit dem Brief würde derjenige als Initiator auftreten, der den Brief bekommen hat. Im Aktivitäten–Manager gibt es einen Menüpunkt, mit dem er das Auftreten des Ereignisses melden kann. Ein Programm kann auch die Rolle des Initiators übernehmen und ein Ereignis melden.

Vor der Bearbeitung einer Aktivität wird überprüft, ob ein in einem Attribut der Aktivität spezifiziertes Ereignis eingetreten ist. Falls das Ereignis noch nicht eingetreten ist, wird die Bearbeitung der Aktivität solange verzögert, bis das Auftreten des Ereignisses gemeldet wird. Falls das Ereignis schon eingetreten ist, wird mit der Bearbeitung der Aktivität fortgefahren. Es kann dann die Vorbedingung der Aktivität geprüft werden. Falls keine Aktivität mehr Interesse an einem Ereignis hat, wird das Ereignis aus der Datenbank gelöscht.

3.1.4. Interne Ereignisse

Als *interne Ereignisse* werden Nachrichten bezeichnet, die zur Steuerung der Workflow–Engine über die zentrale Warteschlange eingesetzt werden. Interne Ereignisse sind häufig Auslöser von Zustandsübergängen bei Aktivitäten, Blöcken und Sphären.

3.2. Ersatzaktivitäten

Ersatzaktivitäten [Sch95] und [SB96] sind ein Konzept, um flexibel auf Fehlersituationen oder zeitliche Engpässe reagieren zu können. Ersatzaktivitäten bzw. Ersatzknoten sind einer normalen Aktivität bzw. Knoten zugeordnet. Die Ausführung von Ersatzaktivitäten wird entweder durch einen Timeout oder durch das Fehlschlagen einer Aktivität ausgelöst. Mit Hilfe der Ersatzaktivitäten kann eine differenzierte Reaktion des Workflowsystems auf den Auslöser erfolgen. Über den timeout kann z. B. eine ausbleibende Bearbeitung einer Aktivität angemahnt werden. Eine Ersatzaktivität wird entweder zusätzlich zur normalen Aktivität gestartet oder die normale Aktivität wird beendet und durch die Ersatzaktivität ersetzt. Der ersetzende Start kann z. B. beim Fehlschlagen einer Aktivität angewendet werden, um eine alternative Lösung des Arbeitsschritts zu ermöglichen. Es kann z. B. auch der Fall auftreten, daß aufgrund der ausgebliebenen Bearbeitung in der Zwischenzeit sich die Vorgehensweise geändert hat. So kann z. B. ein Dokument nicht mehr per Post verschickt werden, sondern muß gefaxt werden. Da der Auslöser für die Ersatzaktivität mehrfach auftreten kann, kann über das Attribut `replaceMode` eingestellt werden, ob der zusätzliche Start einer Ersatzaktivität einmal oder mehrmal durchgeführt werden darf. Ersatzaktivitäten müssen dieselben Input- und Outputcontainer wie die normalen Aktivitäten besitzen, da sie die Stelle der normalen Aktivitäten einnehmen, wenn sie gestartet werden.

Ersatzaktivitäten sind momentan (Stand Ende 1996) noch nicht vollständig implementiert.

3.3. Programm–Pool

Eine ausführlichere Beschreibung findet sich in [Ros96].

Workflow–Management–Systeme werden typischerweise in heterogenen Systemumgebungen eingesetzt, d. h. die Benutzer arbeiten auf unterschiedlichen Rechnerarchitekturen und Betriebssystemen. Diese Heterogenität darf die Funktionalität des Workflowsystems nicht beeinträchtigen. Daher müssen die Applikationen, die zur Bearbeitung von Aktivitäten verwendet werden, für die unterschiedlichsten Rechnersystemen zur Verfügung stehen. Oft ist es jedoch nicht möglich, ein und dieselbe Applikation auf verschiedenen Plattformen zu erwerben bzw. zu erstellen. In einem

solchen Fall müssen Ersatzapplikationen gefunden werden, um die problemlose Abwicklung der Geschäftsprozesse zu gewährleisten.

Der Bearbeiter eines Geschäftsprozesses steht beim Wechsel seines Arbeitsplatzes vor dem Problem, die ihm zugeteilten Aktivitäten mit einer ungewohnten Applikation zu bearbeiten, weil seine gewohnte Applikation auf dem aktuellen Rechnersystem nicht zur Verfügung steht. Außerdem muß bei der Prozeßspezifikation die Existenz der Anwendungen für alle möglichen Plattformen sichergestellt werden, was die Prozeßspezifikation komplexer werden läßt.

Eine Möglichkeit zur Lösung dieser Probleme besteht darin, plattformunabhängige Anwendungsprogramme zu verwenden, die auf allen Arbeitsplatzrechnern eingesetzt werden können. Diese Anwendungsprogramme können bei Bedarf von einer zentraler Stelle aus auf die Arbeitsplatzrechner kopiert werden, was die individuelle Installation auf jedem Rechner erspart.

3.3.1. Probleme bei der Einbindung von Anwendungen in ein Workflow–Management–System

Dieser Abschnitt beschreibt Probleme, die bei der Ausführung von Anwendungsprogrammen in Workflowsystemen entstehen. Die Probleme werden aus der Sicht der Prozeßspezifikation, der Prozeßbearbeitung und der Anwendungsentwicklung beschrieben. Die Probleme rühren meist aus der Heterogenität der verwendeten Rechnersysteme her. Heterogene Rechnerstrukturen sind meist historisch gewachsen. Ein Unternehmen vermehrt mit der Zeit ihre Rechnerausstattung und schafft bestimmte Rechnerarten für spezielle Aufgaben an, die nur mit diesem System zu lösen sind. Dies hat zur Folge, daß in einem Unternehmen Rechner mit unterschiedlicher Hardware und verschiedener Software (Betriebssysteme und Anwendersoftware) existieren. Wenn jetzt ein Workflowsystem zum Einsatz kommt, muß das Unternehmen auf die bestehende informationstechnische Infrastruktur zurückgreifen, weil die Versorgung mit neuen Rechner und der dazugehörigen Software zu kostspielig wäre (Investitionsschutz).

Durch die Heterogenität der Rechnersysteme, die als gegeben angenommen werden muß, entstehen sowohl für die Softwareentwickler (Anwendungsentwicklung, Workflowsystem–Entwicklung, Geschäftsprozeßspezifikation) als auch für die Anwender (Bearbeiter des Geschäftsprozesses) Probleme.

Prozeßspezifikation

Aus der Sicht der Prozeßspezifikation besteht das Problem, daß für die Bearbeitung der gleichen Aktivität verschiedene Anwendungen verwendet werden müssen, wenn die Aufgaben von Benutzern auf verschiedenen Rechnersystemen bearbeitet werden müssen. Normalerweise wird der Bearbeiter dynamisch ermittelt, was zur Folge hat, daß für jede mögliche Plattform eine Anwendung angeführt werden muß. Dies erhöht den Aufwand und die Fehlerwahrscheinlichkeit in der Prozeßspezifikation. Außerdem

muß das Workflow-System selbst je nach Plattform die adäquaten Mechanismen zur Einbindung einer Anwendung verwenden (Aufruf über Kommandozeile, Aufruf einer Funktionsbibliothek usw.). Dies führt dazu, daß die Komplexität des Workflowsystems selbst zunimmt.

Um einen funktionierenden Geschäftsprozeß zu spezifizieren, genügt es nicht, den Aktivitäten die entsprechenden Anwendungen zuzuordnen, es muß auch dafür gesorgt werden, daß die Anwendungen auf allen am Workflowsystem teilnehmenden Arbeitsplatzrechnern installiert sind. Durch die Installation und Wartung der Anwendungen auf jedem Arbeitsplatzrechner entsteht ein nicht zu vernachlässigender Arbeitsaufwand. Je größer die Zahl der verwendeten Rechnersysteme ist, um so größer ist der Aufwand für die Spezifikation eines ausführbaren Geschäftsprozesses, weil für jedes verwendete Rechnersystem die entsprechenden Anwendungsprogramme eingebunden und installiert werden müssen.

Prozeßbearbeitung

Für den Bearbeiter eines Geschäftsprozesses ergeben sich ebenfalls Probleme, die durch die Heterogenität der Rechnerinfrastruktur hervorgerufen werden. Wenn ein Bearbeiter seinen Arbeitsplatz wechselt, weil beispielsweise sein Rechner ausgefallen ist, steht er vor dem Problem, sich in einer völlig anderen Arbeitsumgebung zurechtfinden zu müssen, weil der Ersatzrechner möglicherweise mit einem anderen Betriebsystem arbeitet, das andere Befehlsformate akzeptiert. Außerdem hat der Ersatzrechner möglicherweise eine andere Benutzeroberfläche, mit deren Bedienung der Bearbeiter nicht vertraut ist. Ein wichtiger Aspekt sind in diesem Zusammenhang die Kosten für die Schulung des Mitarbeiters, die notwendig werden können, wenn der Arbeitsplatz gewechselt wird.

Ein weitaus größeres Problem als die ungewohnte Bedienung eines fremden Betriebssystems stellt das Fehlen bestimmter Anwendungsprogramme auf dem Ersatzrechner dar. Der Bearbeiter steht dabei oft vor dem Problem, daß er seine Aufgabe mit einer nicht vertrauten Anwendung erledigen muß, was die Fehlerwahrscheinlichkeit erhöht. Im Extremfall kann der Fall eintreten, daß die benötigte Anwendung auf dem Ersatzrechner überhaupt nicht vorhanden ist und die Arbeit somit nicht erledigt werden kann. Workflowsysteme werden nicht zuletzt deswegen eingesetzt, weil man sich davon eine Beschleunigung der Vorgangsbearbeitung und eine Verringerung der Fehler während der Prozeßbearbeitung verspricht. Diese Vorteile können durch die oben angeführten Probleme zerstört werden, weil ein Prozeßbearbeiter, der in einer ungewohnten Arbeitsumgebung arbeiten muß, fast zwangsläufig mehr Fehler macht. Außerdem wird dadurch auch die Bearbeitungsgeschwindigkeit negativ beeinflußt. Um die optimale Leistung zu erbringen, ist der Mitarbeiter an seinen angestammten Arbeitsplatz gebunden, was nicht immer gewährleistet werden kann.

Ein weiteres Problem sind diejenigen Anwendungsprogramme, die ein Prozeßbearbeiter für die Bearbeitung von Aktivitäten benutzt, die ihm die Wahl des zu benutzenden Werkzeugs freistellen. Hierfür wird er wahrscheinlich seine eigenen spe-

ziellen Programme benutzen, die beim Wechsel des Arbeitsplatzes natürlich nicht mehr zugänglich sind. Durch diese Probleme, die durch die verwendeten Rechnersysteme entstehen, kann die Akzeptanz des Workflowsystems bei den Bearbeitern leiden und sogar dazu führen, daß der Einsatz eines WFMS als hinderlich angesehen wird.

Anwendungsentwicklung

Aus der Sicht des Anwendungsentwicklers besteht das Problem, daß die Anwendungen auf unterschiedliche Plattformen portiert werden müssen. Für Anwendungen, die aus verschiedenen Gründen nicht portiert werden können, müssen Ersatzanwendungen gefunden bzw. implementiert werden, die aber eventuell nicht die volle Funktionalität der Originalanwendung bieten.

Einen anderen Aspekt der Anwendungsentwicklung stellt das Workflowsystem selbst dar, und zwar muß dieses die Möglichkeit bieten, je nach verwendeter Plattform des Prozeßbearbeiters, unterschiedliche Anwendungsprogramme zu benutzen. Dies ist z. B. bei FlowMark der Fall. FlowMark bietet die Möglichkeit, je nach Plattform des Runtime–Clients (Windows, OS/2 oder AIX), unterschiedliche Anwendungen einzubinden. Außerdem bietet FlowMark unterschiedliche Mechanismen für die Einbindung von Anwendungsprogrammen. So können z. B. OS/2–Programme, OS/2–DLL–Funktionen und REXX–Commandfiles ausgeführt werden. Für die Anwendungsentwicklung ergibt sich ein nicht zu vernachlässigender Aufwand, um die angesprochenen Probleme zu lösen. Durch diesen Aufwand entstehen dem Unternehmen zusätzliche Kosten, die die Rentabilität eines Workflowsystems in Frage stellen können.

3.3.2. Ein Lösungskonzept

Dieser Abschnitt stellt die Anforderungen an ein Konzept auf, das in der Lage ist, die oben angeführten Probleme bei der Ausführung von Anwendungsprogrammen in einem Workflowsystem zu lösen. Die wohl wichtigste Anforderung ist, daß ein Prozeßbearbeiter seine Aufgabe unabhängig von dem Rechnersystem, an dem er arbeitet, lösen kann. Dies impliziert, daß die Anwendungsprogramme, die zur Bearbeitung einer Aktivität verwendet werden, unabhängig von der Plattform, auf der sie verwendet werden, immer das gleiche Aussehen und die gleiche Funktionalität bieten (Transparenz der Heterogenität der verwendeten Rechnerinfrastruktur). Das angestrebte Konzept soll die Installation von Anwendungsprogrammen vereinfachen, dabei soll eine Mehrfachinstallation einer Anwendung vermieden werden, so daß eine Anwendung einmal installiert auf allen Arbeitsplatzrechnern eingesetzt werden kann. Das Konzept soll möglichst in Verbindung mit jedem beliebigen Workflowsystem verwendet werden können. Das heißt, daß möglichst keine speziellen Mechanismen eines speziellen Workflowsystem verwendet werden sollen.

Als Lösung dieser Probleme bietet sich an, in Aktivitäten Programme zu verwenden, die auf jeder Plattform ausgeführt werden können. Diese Programme müssen

dazu in einer Programmiersprache implementiert werden, mit der es möglich ist, plattformunabhängige Programme zu erstellen. Typische Vertreter solcher Sprachen sind Interpretersprachen wie Java oder Tcl/Tk. Diese Programme werden zentral in einem Programm–Pool gespeichert. Wenn ein Aktivitäten–Manager ein solches Programm ausführen muß, dann stellt er eine Anfrage an den Programm–Pool und das fragliche Programm wird lokal auf dem Arbeitsplatzrechner des Aktivitäten–Managers installiert.

Die Verwendung von plattformunabhängigen Anwendungsprogrammen beseitigt die Probleme der heterogenen Workflow–Systemumgebung. Die zentrale Verwaltung dieser Programme in einem Programm–Pool vereinfacht den Installations– und Wartungsaufwand für die Anwendungsprogramme des Workflowsystems.

3.4. Ein transaktionales Dateisystem

Im Rahmen von [Täu96] wird ein Konzept für ein Dateiverwaltungssystem als Resource–Manager mit Recovery–Fähigkeit und Dokumentenverwaltungsfunktionalität entworfen. Das zugehörige System ist teilweise realisiert. Dieses Konzept bietet interessante Ansätze für den Einsatz im Dokumentenmanagement und in Workflow–Management–Systemen im allgemeinen und für die Workflow–Transaktionen im besonderen.

3.4.1. Motivation

Das Konzept bezieht seine Motivation zum einen — analog zu den Workflow–Transaktionen — aus der mangelnden Zuverlässigkeit und Fehlertoleranz bestehender Workflow–Management–Systeme, zum anderen aus der Notwendigkeit, strukturierte Dokumente mit umfangreichen Zugriffsmechanismen und -kontrollen als Basis für eine Dokumentenverwaltung anzubieten.

Die Verwaltung von Dokumenten in Multi–User–Entwicklungsumgebungen stellt ein nicht zu unterschätzendes Problem dar. Beispiele für solche Umgebungen sind Software- oder CAD–Entwicklungssysteme. Dabei müssen mehrere Personen, typischerweise quasi gleichzeitig, an einer Vielzahl von Dokumenten arbeiten. Änderungen müssen synchronisiert werden, damit die Konsistenz des Zustands eines Dokumentes gewährleistet werden kann. Die Synchronisation darf aber nicht zu streng sein, damit Parallelarbeit nicht zu sehr eingeschränkt wird. In dieselbe Richtung geht das Granulat der Dokumente, also welche minimale Größe die Dokumente oder Dokumentteile besitzen, auf die sich Synchronisationsmassnahmen beziehen können.

Wenn man bestehende Speicherungssysteme zur Verwaltung von Dokumenten verwenden will, gibt es zwei offensichtliche Ansätze:

- Die Verwendung klassischer Dateisysteme
- Die Verwendung einer (relationalen) Datenbank

Die Verwendung eines klassischen Dateisystems besitzt große Flexibilität, da man relativ viele Freiheiten bei der Verwendung von Speicherungsstrukturen besitzt. Demgegenüber stehen aber hohe Kosten für die Implementierung der notwendigen Mechanismen (Synchronisation, Replikation, Speicherungsstrukturen und allgemeine Zugriffsfunktionen) und eine schwierige Portierung auf andere Umgebungen. Ein transaktionaler Zugriffsschutz ist nur sehr aufwendig zu erreichen.

Bei Verwendung einer relationalen Datenbank sind durch die Abstraktion der Speicherungsdetails und die Abfragesprachen einige der eben angesprochenen Probleme beseitigt. Eine relationale Datenbank ist aber auf die Verwaltung und Bearbeitung großer Mengen von sehr kleinen, ähnlich strukturierten Dateneinheiten mittels kurzer Operationen ausgerichtet, und nicht auf die kooperative Bearbeitung größerer Dokumente über längere Zeit.

3.4.2. Das Konzept

Das *objektstrukturierte Filesystem (ObjFS)* aus [Täu96] ist eine Dateisystemerweiterung mit Funktionalität zur transaktionalen Dokumentenverwaltung. Ein Dokument entspricht dabei in erster Stufe einer Datei, die eine Anwendung verwendet. Diese Dokumente können aber schon auf Ebene des Dateisystems weiter hierarchisch in einzelne *Objekte* strukturiert werden. Es bietet unter anderem folgende Eigenschaften:

- Kompatibilität zum bisherigen Dateisystem
  Der Zugriff auf die Dokumente erfolgt über eine Schnittstelle, die die Standard–Dateisystemschnittstelle als Untermenge beinhaltet. Somit kann bei eingeschränkter Funktionalität auch mit Alt–Software auf die Dokumente zugegriffen werden. Zusätzlich gibt es Zugriffsfunktionen, die die erweiterte Funktionalität (Kontrolle der Transaktionsgrenzen, Verwaltung von Teildokumenten, usw.) anbieten.

- Transaktionaler Schutz mit der Vorbereitung zur Unterstützung abgeschwächter Transaktionsmodelle
  Das Dateisystem ist als Resource–Manager mit OTS–Schnittstelle realisiert und kann somit in Transaktionen eingebunden werden. Es führt einen Log und besitzt eine Sperrverwaltung.

- Synchronisation auf feinem Granulat
  Dokumente können hierarchisch strukturiert werden. Die Objekte, die dabei als kleinste Einheit entstehen, können einzeln mit Sperrmechanismen geschützt werden. Die Größe der Objekte bestimmt die erstellende Anwendung selbst. Sie hat somit Einfluß auf den möglichen Kooperationsgrad.

- Varianten- und Referenzenverwaltung
  Zu jedem Objekt können Varianten erstellt und automatisch verwaltet werden. Über Referenzen kann über Dokumentteile navigiert werden.

3.4.3. Integration in ein WFMS

Das ObjFS läßt sich als reines Dokumentenverwaltungssystem auffassen und in dieser Funktion in ein Workflowsystem einbinden. Dies wird hier nicht weiter betrachtet. Die zweite, schon beim Konzept der Workflow–Transaktionen angesprochene Möglichkeit, besteht darin, den Integrationscharakter des Dateisystems für die Workflow–Transaktionen auszunutzen.

Dadurch daß das transaktionale Dateisystem die Standard–Dateisystemschnittstelle implementiert, kann Alt–Software das Dateisystem zur Speicherung ihrer Dateien verwenden. Über die Resource–Manager Eigenschaft können die Operationen transaktional geschützt werden. Im Falle der Workflow–Transaktionen ist ein Schutz durch klassische Transaktionen mit den ACID–Eigenschaften verlangt. Die Änderungen auf den entsprechenden Dateien werden nach außen isoliert. Ein Rollback macht die Dateiänderungen komplett rückgängig.

Wenn die Alt–Software ihren Zustand komplett auf dem Dateisystem ablegt, kann somit aus Sicht eines Beobachters ein echt transaktionales Verhalten erreicht werden, obwohl die Anwendung nicht darauf vorbereitet ist. Somit kann die Alt–Software auch als ACID–Aktivität sinnvoll in einer Workflow–Transaktion eingesetzt werden. Man muß dabei aber berücksichtigen, daß noch zur Laufzeit der Anwendung ein Rollback erfolgen kann, der ihr quasi ihre bisherige Arbeit „unter den Füßen wegzieht“. Da dies zwangsläufig zu massiven Inkonsistenzen führen würde (Die Anwendung bekommt von einem Rollback vorerst einmal nichts mit), muß dafür gesorgt werden, daß nach einem Rollback keine weiteren Dateizugriffe dieser Anwendung mehr ausgeführt werden. Die Anwendung sollte so schnell wie möglich abgebrochen werden. Wenn die Alt–Anwendung fehlerhaft abbricht, ist dies ein Grund, die Transaktion abzubrechen. Dies muß vom System erkannt werden können.

Aus Sicht des Workflowsystems müssen spezielle Vorkehrungen getroffen werden, um eine solche Integration einer nicht–transaktionalen Aktivität in eine Sphäre zu erreichen. Nur die Workflow–Engine kennt den Transaktionskontext, mit dem die Sphäre geschützt wird. Diesen Kontext muß das transaktionale Dateisystem für die Ausführung der Dateioperationen der Anwendung verwenden. Normalerweise gibt der aufrufende Client (in diesem Fall die Alt–Anwendung) den Transaktionskontext implizit oder explizit beim Aufruf mit. Dies ist in diesem Fall nicht möglich, da die Anwendung kein Wissen über Transaktionen besitzt. Deshalb muß das transaktionale Dateisystem eine Abbildungsfunktion besitzen, die den aufrufenden Betriebssystemprozeß der Alt–Anwendung dem korrekten Transaktionskontext zuordnet. Um wiederum diese Funktion realisieren zu können, muß das Workflowsystem dem transaktionalen Dateisystem jeweils mitteilen, welcher Anwendungsprozeß welchen Transaktionskontext benötigt.

Das ObjFS ist in Grundzügen auf OS/2 als „installable Filesystem“ implementiert. Die Integration in das Workflowsystem Surro ist bisher nicht erfolgt.

4. Der Aufbau des Workflowsystems Surro

4.1. Motivation des Prototypen

In diesem Abschnitt soll kurz dargelegt werden, aus welchen Gründen und mit welchen Zielen im Rahmen dieses Projekts ein Prototyp eines Workflow–Management–Systems erstellt worden ist.

- Erstes Ziel des Prototypen ist der Nachweis der Realisierbarkeit der in Abschnitt 2 beschriebenen Konzepte zur Fehlertoleranz. Dabei stehen an erster Stelle die Workflow–Transaktionen und an zweiter Stelle die Kompensations–Transaktionen. Es soll dabei der Aspekt des Realisierungsaufwandes betrachtet und beurteilt werden. Es soll herausgefunden werden, ob sich eine Kombination des Transaktionsdienstes *Object Transaction Service (OTS)* [OTS94], der CORBA–Implementierung DSOM und der Datenbank DB2 für die Realisierung geeignet. Es soll überprüft werden, inwieweit sich Sperren auf den Workflow–Verwaltungsdaten bei der Abarbeitung von Sphären auswirken. Kann es an den Grenzen von Sphären zu Problemen kommen?

- Ein weiteres Hauptziel ist das Sammeln von praktischer Erfahrung im Umgang mit dem Transaktionssystem OTS als Bestandteil der CORBA–Implementierung DSOM 3.0 von IBM. Dabei steht die Zusammenarbeit von OTS als externer Transaktionsmanager mit der Datenbank DB2 im Vordergrund, da hier aufgrund des Beta–Stadiums der Version 3.0 von DSOM nicht auf Erfahrungen und auch nicht auf vorhandene Dokumentation zurückgegriffen werden kann. Der allgemeine Umgang mit CORBA als Programmierplattform ist ebenfalls ein wichtiger Punkt.

- Der Prototyp soll zukünftig als Experimentierplattform für weitere Konzepte dienen, beispielsweise Ad–hoc–Modifikationen von Workflows, Ereignisverwaltung oder die Integration von Componentware–Systemen. Der Prototyp soll auch als Vorbereitung auf künftige Objekttechnologien, die in Aktivitäten eingesetzt werden, verstanden werden.

4.1.1. Vorgeschichte und Entstehung

Zum Verständnis einiger strukturell relevanter Entscheidungen bezüglich des Prototypen ist ein kurzer Abriß der Entstehungsgeschichte sinnvoll.

Das im Prototyp Surro[1] verwendete Workflow–Modell orientiert sich stark am Workflow–Modell des IBM–Produkts FlowMark. Dies soll die Übertragbarkeit der Ergebnisse aus der Prototypentwicklung auf FlowMark ermöglichen. Aus dem gleichen Grund wurde auch teilweise die Terminologie von FlowMark (z. B. PEC) übernommen.

Als Ausgangspunkt für die Surro Prototypentwicklung wurde ein an der Universität Stuttgart erstelltes Workflowsystem verwendet, das im Rahmen einer Diplomarbeit entstanden ist. Konzepte und Programmcode konnten so wiederverwertet werden. Eine ausführliche Beschreibung dieses Systems ist in [Sch95] zu finden. Dieses Ausgangssystem wurde so weit modifiziert, daß es Grundkonzepte verwendet, die zu FlowMark weitgehend identisch sind. Das Ausgangssystem, und somit auch die momentane Version der Engine, ist in der Programmiersprache Tcl/Tk [Ous94] implementiert, die sich gut zum Erstellen von Prototypen mitsamt der grafischen Benutzungsoberfläche eignet. Aufgrund dieser Eigenschaft sind auch andere, später dazugekommene Komponenten in dieser Sprache implementiert (z. B. der Monitor). Der Aktivitäten–Manager und der Programm–Pool sind im Rahmen einer Studienarbeit in Java realisiert.

Aufgrund des Projektziels, den Transaktionsverwalter OTS aus der DSOM–CORBA–Implementierung von IBM für den Einsatz bei der Realisierung von Workflow–Transaktionen zu evaluieren, und aufgrund der Tatsache, das OTS momentan nur auf OS/2 in einer Betaversion verfügbar ist, mußte der gesamte Teil der Engine, der die Dienste von DSOM und OTS nutzt, auf OS/2 implementiert werden. Die Implementierungssprachen auf der OS/2–Plattform sind C und C++. Dies führt zu dem im folgenden Abschnitt beschriebenen strukturellen Aufbau des Surro Prototypen.

4.2. Aufbau des Surro Prototypen

In diesem Abschnitt werden die Entstehung, Eigenschaften und die Struktur des Prototypen erläutert. In Abb. 4.1 ist ein Übersichtsbild über die Struktur des entwickelten Prototypen zu sehen.

Der Prototyp zeigt eine deutlich zweigeteilte Struktur. Die Aufteilung auf zwei Betriebssysteme ist historisch bedingt und aus vorgegebenen Randbedingungen zu erklären. Der Prototyp existiert in zwei Varianten. Zum einen gibt es die Möglichkeit, das System ohne Einbeziehung der OS/2–Seite zu betreiben. Dabei besteht die Einschränkung, daß für die Realisierung der Sphären keine echten Transaktionen zur Verfügung stehen. Die in dieser rein UNIX–basierten Variante verwendete Datenbank

[1] Surro steht für Surrogat

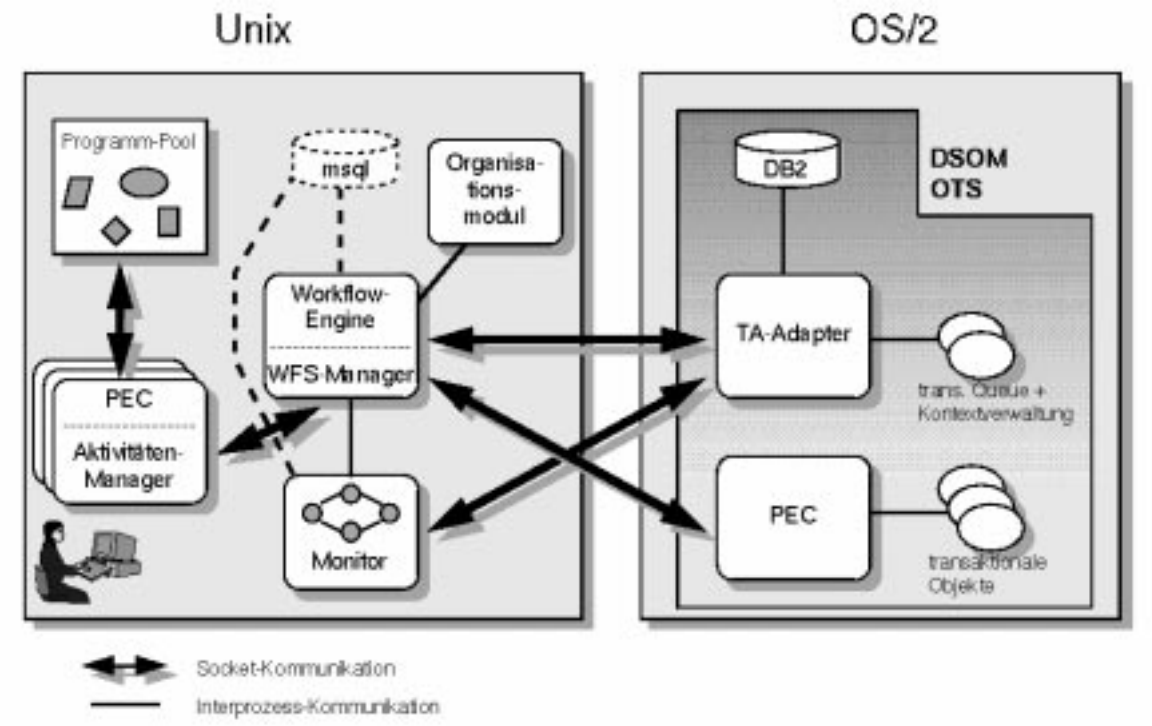

Abbildung 4.1.: Struktureller Aufbau des Surro Prototypen

mSQL [Hug96] ist nicht transaktionsfähig, ebensowenig wie die auf dieser Plattform vorhandenen Aktivitätenprogramme. Diese Variante ist zum Test der Grundfunktionalität des Workflowsystems gut geeignet.

Involviert man die OS/2–basierten Komponenten, fällt die Verwendung der mSQL–Datenbank weg. Dafür wird die transaktionsfähige DB2 herangezogen. Über OTS als globalen Protokoll–Koordinator ist dann echtes Transaktions–Management möglich. Ebenso sind transaktionale Aktivitäten vorhanden.

4.2.1. Aufgaben der Systemkomponenten

Die nähere Funktionsweise und Implementierungsdetails des Gesamtsystems werden im Kapitel 6 erläutert. In diesem Abschnitt folgt eine Beschreibung der Aufgaben der einzelnen Systemkomponenten.

Workflow–Engine

Die zentrale Komponente des Systems ist die *Workflow–Engine*. Sie hat die Aufgabe, den Ablauf der Prozesse zu steuern und zu überwachen. Sie ermittelt anhand der ihr zur Verfügung stehenden Informationen die nächsten zu unternehmenden Arbeitsschritte (Aktivitäten). Dabei kontrolliert sie die Zustandsänderungen der laufenden Prozesse und Aktivitäten. An Informationen verwendet die Engine die statischen Prozeß– und Ressourcendefinitionen in der Workflow–Datenbank und die Meldun-

gen der anderen Komponenten des Systems über eingetretene Ereignisse. Sie schickt Informationen über zu bearbeitende Aktivitäten an die Aktivitäten–Manager. Sie reagiert auf Beendigungen von Aktivitäten durch die Bearbeiter und kontrolliert dabei die Zustandsänderungen. Sie informiert den Monitor über Zustandsänderungen. Sie führt Datenbankzugriffe im Auftrag für andere Systemkomponenten durch.

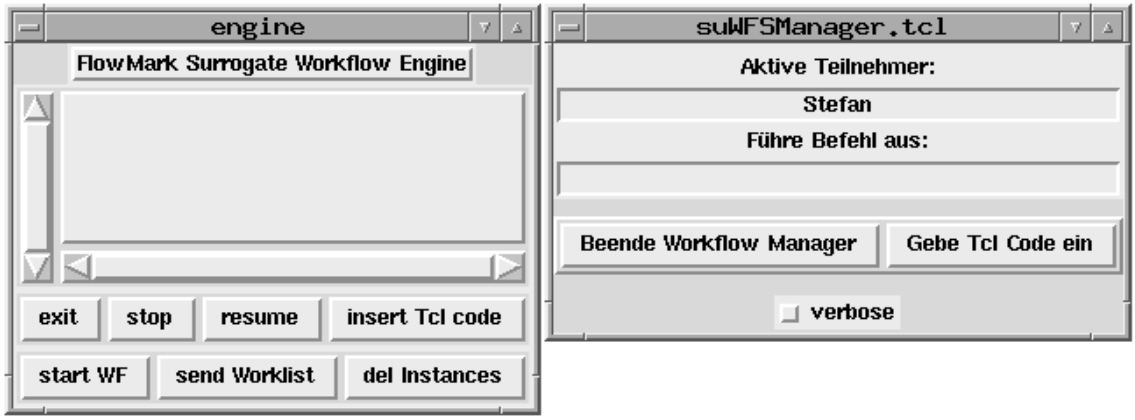

Abbildung 4.2.: Oberfläche der Engine und des WFS–Managers

Workflow–Session–Manager

Der *Workflow–Session–Manager* (WFS–Manager) ist im Prototyp ein eigenständiger Prozeß, hat aber eine sehr enge Bindung zur Engine. Aus diesem Grund ist er in Abb. 4.1 nicht als eigenständiges Objekt dargestellt. Er ist für die Kommunikation mit den Benutzerkomponenten (Aktivitäten–Manager, PEC) und für die Verwaltung der Session–Daten der Benutzer zuständig. Er verfügt über die Informationen, wer momentan im System angemeldet ist.

Die Benutzungsoberflächen von Engine und WFS–Manager (Abb. 4.2) haben vorerst reine Debug–Funktionalität.

Datenbank (mSQL / DB2)

Die Datenbank speichert zum einen statische Informationen, d. h. die Definitionen der verfügbaren Prozesse (Aktivitäten, Kontrollfluß, Datenfluß, Datencontainer, usw.) und das Organisationsmodell. Zum anderen beinhaltet sie dynamische Informationen in Form des momentanen Zustands jedes derzeit ablaufenden Prozesses. Die Engine sichert jede Zustandsänderung eines Prozesses sofort in der Datenbank.

Der Prototyp hat die Möglichkeit, alternativ die *mSQL*–Datenbank auf UNIX–Seite oder die *DB2*–Datenbank auf OS/2–Seite zu verwenden. Da nur die DB2–Datenbank Transaktionen beherrscht, ist bei Verwendung der mSQL–Datenbank die Transaktionseigenschaft der Sphären nicht gegeben.

TA–Adapter, transaktionale Queue, Kontextverwaltung

Der *Transaktions–Adapter* ist der eigentliche Client zur DB2–Datenbank. Nur er ruft, stellvertretend für die Engine, das API der DB2 auf. Daneben ist seine Hauptaufgabe die Verwaltung der Transaktionskontexte und einer transaktionalen Message–Queue (jeweils in einem eigenen DSOM–Objekt). Die *transaktionale Queue* wird von der Engine zur sicheren Zwischenspeicherung von internen Nachrichten verwendet. Der Adapter ist der Prozeß, der die Transaktions–Management–Objekte, die OTS definiert (Coordinator, Terminator, usw.), erzeugt und aufruft.

Der TA–Adapter gehört logisch zur Engine, muß aber als getrennter Prozeß auf der OS/2–Seite implementiert werden, da die Engine keine Möglichkeit hat, DSOM 3.0 und damit OTS zu verwenden. Der Adapter bietet der Engine eine Schnittstelle zum Aufruf der Datenbankfunktionen auf die DB2 und des Transaktionsmanagements an.

Aktivitäten–Manager

Der Aktivitäten–Manager [Ros96] ist die Schnittstelle des Systems zum normalen Benutzer. Der Aktivitäten–Manager besitzt eine graphische Benutzungsoberfläche (siehe Abb. 4.3), die die *Arbeitsliste* zeigt. Der Benutzer erhält über den Aktivitäten–Manager Informationen über die anstehenden Arbeiten (oberste Liste), die momentan in Bearbeitung befindlichen (mittlere Liste) und die unterbrochenen Aktivitäten (untere Liste, leer). Er hat unter anderem folgende Interaktionsmöglichkeiten:

- Starten, Unterbrechen, Wiederaufnehmen, Auslassen von Aktivitäten
- Starten von neuen Prozessen
- Meldung von externen Ereignissen
- Aufruf des Vorgangsinformationssystems (bzw. des Monitors)

PEC

Der *Program Execution Client* (PEC) ist ein Programm, das für die Ausführung und Überwachung von Aktivitätenprogrammen zuständig ist. Er bietet den Aktivitäten ein API an, das die Abfrage von Prozeßinformationen durch das Aktivitätenprogramm erlaubt, z. B. den Inhalt von Datencontainern, und das Beschreiben der Daten–Outputcontainer ermöglicht.

Auf UNIX–Seite ist der PEC in den Aktivitäten–Manager integriert. Dort hat er auch zusätzlich die Aufgabe, mit dem Programm–Pool zu kommunizieren (siehe dort). Auf OS/2–Seite gibt es zur Zeit keinen Aktivitäten–Manager mit Benutzungsoberfläche. Der OS/2–PEC hat allein die Aufgabe, transaktionale Sphären–Aktivitäten automatisch auszuführen. Die Durchführung einer transaktionalen Aktivität besteht aus dem Aufruf einer Methode eines transaktionalen Objekts im richtigen Transaktions–Kontext.

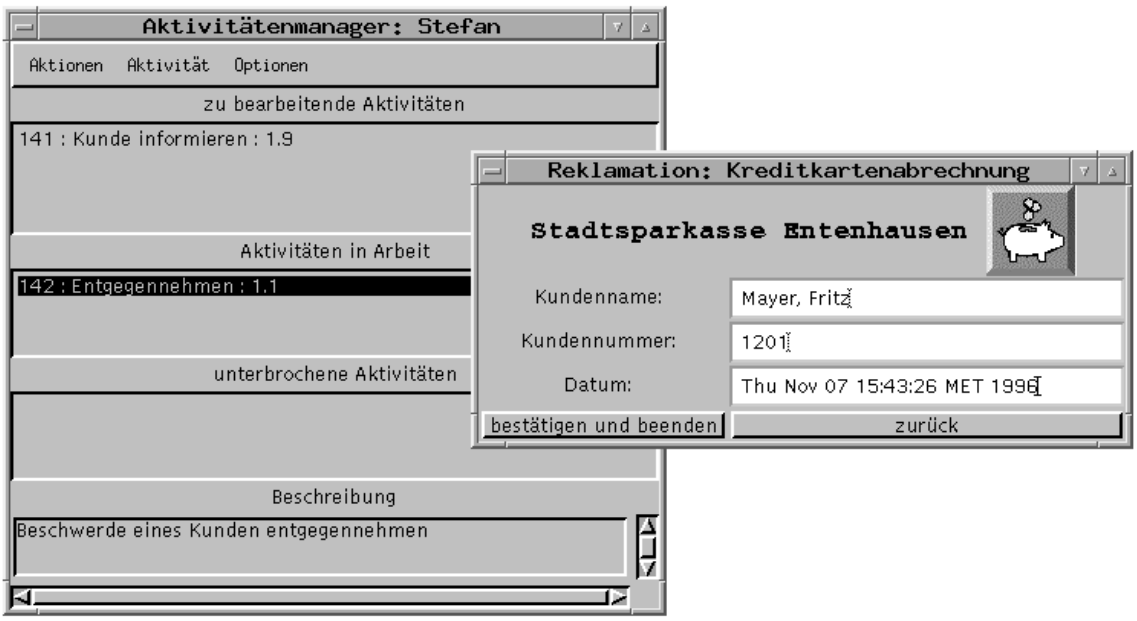

Abbildung 4.3.: Aktivitäten–Manager und eine Aktivität

Transaktionale Objekte

Transaktionale Objekte sind DSOM–Objekte, die ein OTS–konforme Schnittstelle besitzen und somit in einen Transaktionskontext eingebunden werden können. Methodenaufrufe auf diese Objekte werden, vorausgesetzt der Aufruf erfolgt in einem entsprechenden Kontext, transaktional geschützt. Ein derartiger Methodenaufruf ist eine transaktionale Aktivität und kann innerhalb von Workflow–Transaktionen ausgeführt werden.

Programm-Pool

Der Programm–Pool stellt an zentraler Stelle Programme zur Verfügung, die zur Ausführung von Aktivitäten benötigt werden. Auf Anfrage vom PEC installiert der Programm–Pool–Manager Programme zur Ausführung einer Aktivität auf einer Maschine, die noch keine lokale Kopie des Programms besitzt. Der Programm–Pool berücksichtigt dabei verschiedene Plattformen und kann auch Programme anhand von bestimmten Attributen auswählen (Beispielanfrage: Liefere mir ein Programm zum Anzeigen einer MS Word-Datei für eine Windows 3.x–Plattform). Über diese Verteilungsmethode können auch plattformunhängige Programme verteilt werden, damit wird der Anschluß eines neuen Rechners am Workflowsystem vereinfacht, da die ganzen Installationsarbeiten nicht nötig sind.

Monitor

Der Monitor ist die bisher einzige Komponente des Vorgangsinformationssystems. Er zeigt Prozeßdefinitionen (die Templates) und den momentanen Zustand laufender Prozesse (Instanzen) graphisch an (siehe Abb. 4.4). Über den Monitor können auch

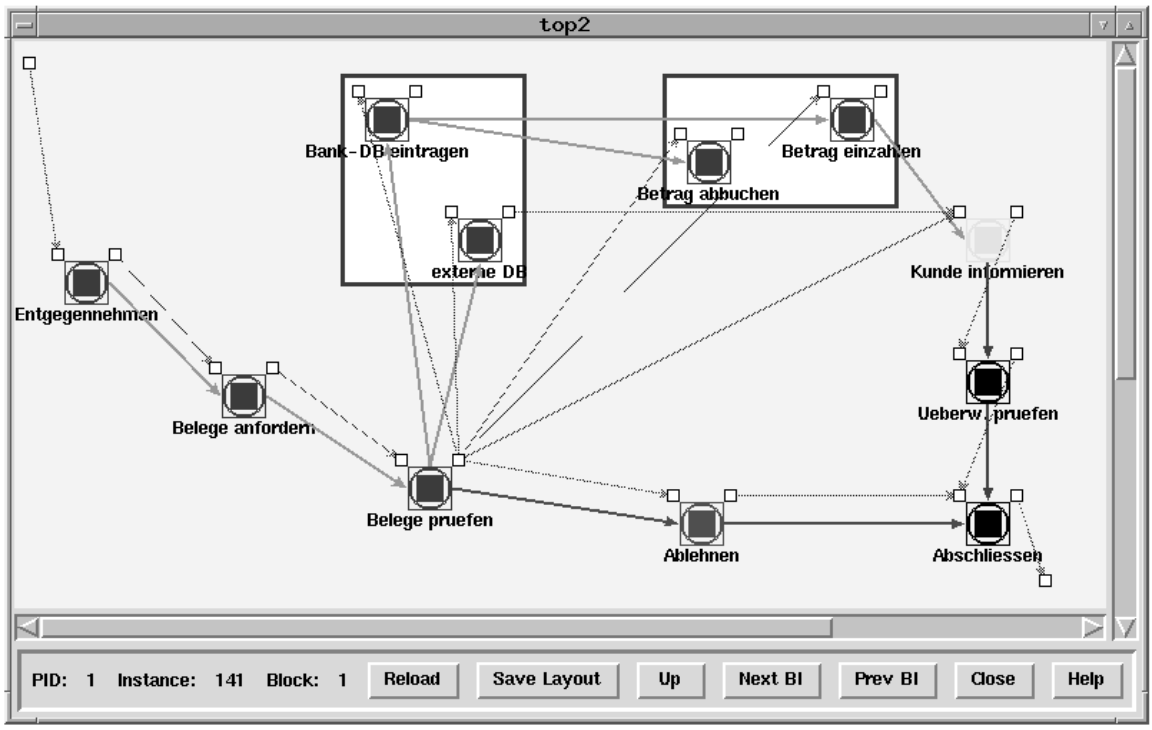

Abbildung 4.4.: Workflow–Monitor mit Darstellung des Beispielprozesses aus A.1

Detailinformationen über die Bestandteile und den Zustand des Prozesses abgefragt werden. Der Monitor holt sich die benötigten strukturellen Informationen direkt aus der Datenbank. Zusätzlich erhält er von der Engine Nachrichten über Zustandsänderungen der Aktivitäten und Konnektoren.

Organisationsmodul

Das Organisationsmodul ist für die Verwaltung der zugrundeliegenden Aufbauorganisation und für die Auswahl geeigneter Bearbeiter (allgemeiner: Ressourcen) für die Ausführung einer Aktivität zuständig. Die Aufbauorganisation spiegelt die Struktur des Unternehmens bzw. der Organisation wieder, die das Workflowsystem einsetzt. Relevant für ein Workflowsystem sind Informationen über die Ressourcen (Personen, Standorte, Maschinen) und deren Eigenschaften und Fähigkeiten. Dazu kommen Beziehungen zwischen den Ressourcen, wie Untergebenenverhältnisse zwischen Personen und Mengen von Personen mit ähnlichen Fähigkeiten (Rollen).

Das Organisationsmodul erhält von der Engine Anfragen nach Ressourcen mit bestimmten Auswahlkriterien und liefert einen oder mehrere in Frage kommenden Bearbeiter zurück. Dabei kann das Modul noch Informationen von der Engine erfragen, typischerweise bezüglich der Historie des Prozesses.

Grafischer Workflow–Editor
Diese Systemkomponente wurde bisher [2] nicht implementiert. Ihre Aufgabe ist die interaktive grafische Erstellung von Geschäftsprozeß–Spezifikationen, die Modellierung der Aufbauorganisation und die Speicherung der entsprechenden Daten in der Datenbank. Alle bisher implementierten Geschäftsprozeß–Modelle sind bisher direkt in SQL angegeben.

4.3. Kommunikation

Die einzelnen Komponenten kommunizieren untereinander über festgelegte Schnittstellen. Aufgrund der Entscheidung für die Weiterverwendung des bestehenden Prototypen aus [Sch95] und der durch IBM vorgegebenen Verwendung von OTS ist es leider nicht möglich bzw. sinnvoll, eine einheitliche Kommunikationsart zu verwenden. Anzustreben wäre, die gesamte Kommunikation über den CORBA Methodenaufruf (bzw. einen RPC–Mechanismus) abzuwickeln, was aber daran scheitert, daß die für OTS benötigte Version 3.0 von DSOM noch nicht auf UNIX verfügbar ist. Zum anderen existieren keine Tcl/Tk–Schnittstellen zu DSOM. Aus diesem Grund wird auf die nächsttiefere Schicht zurückgegriffen, die allen Systemkomponenten zur Verfügung steht: TCP/IP–Sockets. Zusätzlich wird die von Tcl/Tk angebotene, auf eine Maschine beschränkte Interprozeß–Kommunikation (IPC) verwendet, da dieser Mechanismus äußerst einfach zu verwenden ist. Dies führt zu folgender Kommunikationsstruktur:

In Abb. 4.1 wird unterschieden zwischen (TCP/IP-)Socket- und Interprozeß–Kommunikation. Zwischen UNIX und OS/2 wird über TCP/IP–Sockets kommuniziert, da zwangsläufig verschiedene Maschinen involviert sind. Dies betrifft die Kommunikation zwischen Engine und TA–Adapter, zwischen WFS–Manager und dem OS/2–PEC und zwischen dem Monitor und dem TA–Adapter. Zwischen WFS–Manager und Aktivitäten–Manager werden ebenfalls Sockets verwendet, da die Aktivitäten–Manager prinzipiell auf beliebigen Maschinen ablaufen können. Analoges gilt für den Programm–Pool.

Zwischen Prozessen auf der UNIX–Seite, die in Tcl/Tk implementiert sind, wird der Tk-Befehl `send` verwendet, der äußerst einfach zu verwenden ist. Mit `send` kann man an einen anderen Tcl/Tk–Interpreter einen beliebigen Befehlsstring senden, den dieser dann ausführt[3]. Mit diesem Befehl kann eine Art Remote Procedure Call (RPC) realisiert werden. Diese RPC–Kommunikation wird zwischen der Engine und dem WFS–Manager und zwischen der Engine und dem Monitor verwendet. Das hat die Auswirkung, daß diese Prozesse auf derselben Maschine unter demselben X–Server ablaufen müssen.

[2] Stand 1996
[3] Die Sicherheitsprobleme, die daraus entstehen, und wie sie behandelt werden, wird hier nicht näher dargestellt

Die Kommunikation zwischen der Engine und der mSQL–Datenbank erfolgt über das API, das mSQL zur Verfügung stellt. Auf der OS/2–Seite erfolgt die Kommunikation entweder über vorgegebene API's (zwischen dem TA-Adapter und der DB2 wird das DB2 Call Level Interface CLI benutzt) oder über den objektorientierten RPC, den DSOM als CORBA–Implementierung zur Verfügung stellt. Alle übrige Kommunikation auf der OS/2–Seite verwendet diesen Mechanismus. Die CORBA–Kommunikation ist prinzipiell ortstransparent. Die Komponenten (z. B. die transaktionalen Objekte) könnten auch auf verschiedenen Maschinen ablaufen, ohne daß die Implementierung geändert werden müßte.

4.3.1. Die Schnittstellen zwischen den Systemkomponenten

Im folgenden werden kurz die wichtigsten Schnittstellen erläutert, die im Rahmen des Prototypen definiert und/oder verwendet werden.

mSQL–Datenbank

Die mSQL–Datenbank bietet ein C–API an, mit dessen Hilfe ein Tcl–Interface realisiert ist.

TA–Adapter

Der TA–Adapter bietet der Engine (und dem Monitor) eine Schnittstelle an, die weitgehend identisch zu der Schnittstelle ist, die die mSQL–Datenbank gegenüber der Programmiersprache Tcl anbietet. Der Vorteil dieser Vorgehensweise ist, daß man ohne großen Programmieraufwand zwischen der mSQL- und der DB2–Datenbank wechseln kann.

Die Schnittstelle bietet Aufrufe zur Ausführung von SQL–Befehlen, zum Abfragen von Ergebnismengen von SELECT–Statements, zum Abfragen von Informationen über Relationen, zum Verwalten mehrerer logischer Verbindungen analog zur mSQL–Schnittstelle. Dazu kommen spezielle Funktionen zur Transaktionsverwaltung, die in der mSQL–Schnittstelle nicht vorhanden sind. Dies sind Aufrufe zum Begin, Commit und Rollback einer Transaktion und zum Verändern des Transaktionskontextes.

Dazu bietet der TA–Adapter Aufrufe zur transaktionalen Queue an, also unter anderem zum Einfügen eines Elements (Enqueue), Abholen (Dequeue) und Abfrage der Anzahl der Elemente in der Queue.

DB2 $\Longleftrightarrow$ TA–Adapter

Zum Zugriff auf die DB2 wird das Call Level Interface (CLI) verwendet. Zur Anbindung des externen Transaktionsmanagers wird das X/Open XA–Interface [X/O91] herangezogen. DSOM OTS besitzt Vorkehrungen zur Verwendung dieser Schnittstelle.

WFS–Manager $\Longleftrightarrow$ Aktivitäten–Manager und PEC

Zwischen diesen Komponenten wird ein API definiert, das sich stark an das WAPI–Interface (Interface 2) der WfMC–Interface–Spezifikation anlehnt. Hier werden Funk-

tionen zum An- und Abmelden vom System, Übergeben von Aktivitäten, Beenden von Aktivitäten, Abfragen von Datencontainern usw. definiert. Eine ausführliche Beschreibung findet sich in [Ros96].

Zwischen dem OS/2–PEC und dem WFS–Manager wird nur ein kleiner Ausschnitt des API's benötigt, da der PEC noch keine Benutzerinteraktion kennt.

Workflow–Engine $\Longleftrightarrow$ Monitor
Diese Schnittstelle beschränkt sich auf den RPC–Aufruf einer einzigen Funktion des Monitors durch die Engine, wobei dem Monitor die Zustandsänderung eines Prozesses mitgeteilt wird. Der Monitor aktualisiert daraufhin seine Anzeige.

PEC $\Longleftrightarrow$ Programm–Pool
Der Programm–Pool stellt ein Interface zur Verfügung, das unter anderem Funktionen zur Auswahl von Programmen anhand bestimmter Kriterien, zum Abfragen von Attributen und zum Anfordern von Programmen enthält, die dann lokal installiert werden.

Workflow–Engine $\Longleftrightarrow$ Organisationsmodul
Das Organisationsmodul stellt zur Zeit eine RPC–Funktion zur Verfügung, die anhand der mitgelieferten Ressourcenbeschreibung einen geeigneten Bearbeiter zurückliefert. Dabei kann z.B. unter anderem noch angegeben werden, ob Vertretungsregelungen berücksichtigt werden sollen oder nicht. Ein Zugriff auf die Benutzer–Verwaltungsdaten des Workflow–Session–Manager durch das Organisationsmodul ist möglich.

5. Das Datenmodell von Surro

In diesem Kapitel wird das dem Workflow–Management–System Surro zugrundeliegende Datenmodell beschrieben. Der Aufbau des Datenmodells, die die Prozeßdefinitionen und das Organisationsmodell beinhalten, wird anhand eines Entity–Relationship–Diagramms erläutert. Danach werden alle Relationen im einzelnen aufgeführt und dokumentiert.

5.1. Begriffe

Die Relationen können in vier Klassen eingeteilt werden. In der Klasse der **Template–Relationen** sind alle Relationen zu finden, die für die Spezifikation der Geschäftsprozesse benötigt werden. Ein großes T am Ende des Namens einer Relation kennzeichnet diese Klasse. Die Daten in den Template–Relationen müssen durch den Workflow–Editor bei der Modellierung eines Geschäftsprozesses erzeugt werden. Die Klasse der **Instanz–Relationen** besitzt dieselben Relationen wie die Template–Klasse, wobei das T am Ende des Namens fehlt. Bei der Instanziierung eines Workflows werden alle Daten des Prozesses aus den Template–Relationen in die entsprechenden Instanz–Relationen kopiert. Zusätzlich besitzen die Instanz–Relationen weitere Schlüsselattribute, die die Instanz eindeutig identifizieren und die nur in Workflow–Instanzen benötigt werden. Die dritte Klasse von Relationen (**Organisations–Relationen**) werden für die Beschreibung der Aufbauorganisation des Unternehmens oder der Behörde benötigt. In der vierte Klassen sind Relationen für die Verwaltung von internen Workflowdaten, z. B. Instanz–Zähler (**Verwaltungs–Relationen**).

Der Begriff **Knoten** wird als Generalisierung für eine Aktivität, einen Block oder ein Prozeß bzw. Subprozeß benutzt. Bei der Modellierung eines Geschäftsprozesses als Graph treten diese Objekte als Knoten auf, während Daten– und Kontrollflußkonnektoren die Kanten repräsentieren.

Als **Prozeßebene** wird in diesem Zusammenhang eine Menge von Knoten bezeichnet, die auf einer Ebene im Geschäftsprozeß–Modell liegen. Auf der obersten (der Toplevel–) Ebene 0 liegt ein einziger Prozeßknoten. Die Realisierung dieses Prozesses findet sich auf der darunterliegenden Prozeßebene 1. Die Realisierung des Blockes findet sich auf Prozeßebene 2 (siehe Abbildung 5.1). Auf diese Weise wird eine Hierarchie von Prozeßebenen gebildet, die in ihrer Gesamtheit einen Geschäftsprozeß

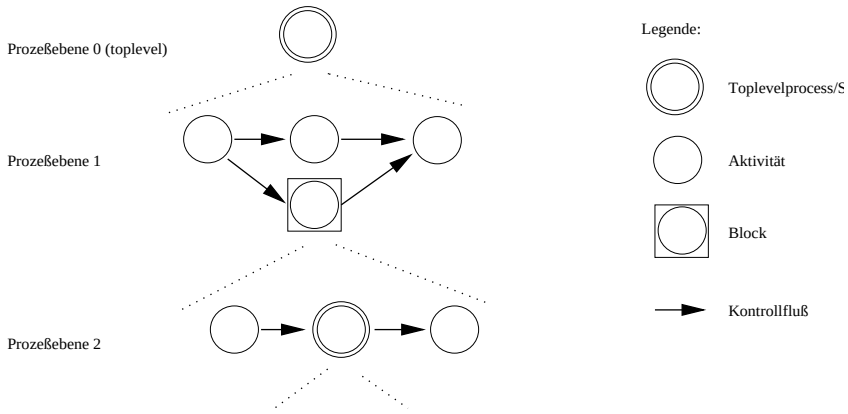

Abbildung 5.1.: Die Prozeßebenen bei der Geschäftsprozeß–Beschreibung

modellieren.

5.2. Instanziierung von Workflows

Die Instanziierung eines Workflows wird durch die interne Nachricht `startWorkflow` ausgelöst. Daraufhin wird der toplevel–Prozeßknoten instanziiert. Mit diesem Knoten wird auch die oberste Prozeßebene instanziiert, die über das Attribut `processID` identifizierbar ist. Die jeweils in Blöcken oder Unterprozessen enthaltenen nächsten Prozeßebenen werden erst dann instanziiert, wenn der Kontrollfluß den Knoten des Block oder des Prozesses erreicht. Eine Prozeßebene wird workflow–lokal über eine `blockInstanceID` identifiziert, d. h. die blockInstanceID beginnt in jedem Workflow mit 1.

Wenn am Ende eines Blocks über das Attribut `exitcondition` festgestellt wird, daß ein weiterer Durchlauf des Blocks nötig ist, dann wird die Prozeßebene des Blocks neu instanziiert. Ein Block wird auf diese Weise als ein Schleifenkonstrukt benutzt.

Nach der Instanziierung eines Workflows kann dieser durch ein geeignetes Modifikationsmodul jederzeit in den Instanzrelationen geändert werden. Dieses Modul muß dafür sorgen, daß nur konsistente Änderungen zur Ad–hoc–Modifikationen des Workflows durchgeführt werden. Dieses Modul ist aber bisher noch nicht implementiert (Stand Ende 1996).

5.3. Das ER–Modell

In den Abbildungen 5.2 und 5.3 sind die Klassen der Template–Relationen und der Organisations–Relationen als ER–Diagramme dargestellt. Die Instanzrelationen werden nicht dargestellt, da diese sich von den Templates nur durch zusätzliche Primärschlüsselattribute unterscheiden. Auf die Darstellung der Nicht–Schlüssel–

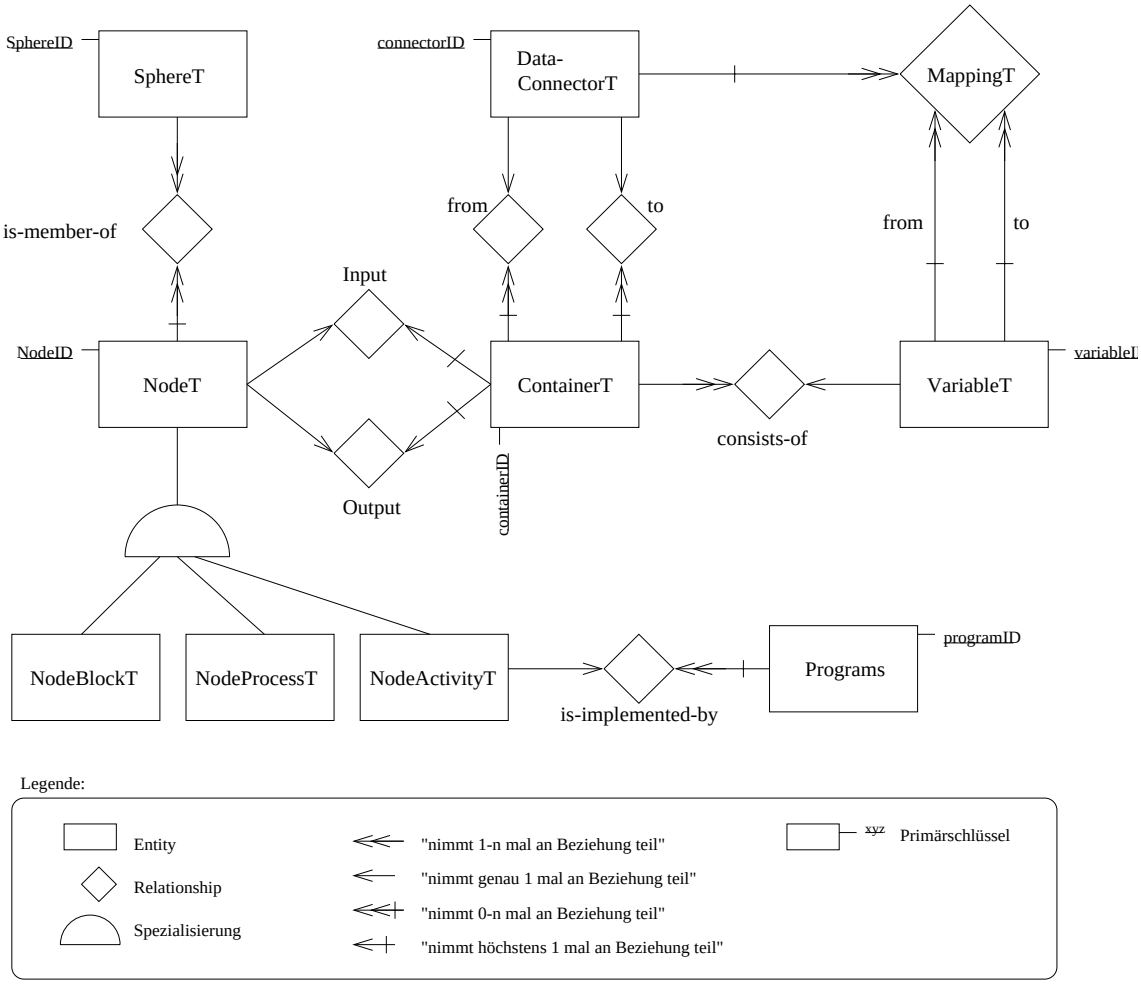

Abbildung 5.2.: ER–Diagramm des Prozeßmodells

Attribute wird aus Platzgründen verzichtet. Diese Attribute sind in der Beschreibungen der Relationen zu finden.

Bei einer direkten Umsetzung des ER–Diagramms (Abb. 5.2) in ein Relationenschema müßte eine Relation mit Namen `VariableT` entstehen. Um zu einer Vereinfachung des Relationenschemas zu kommen wurden die Relationen `VariableT` und `ContainerT` zu einer einzigen zusammengefaßt, die `ContainerT` heißt.

Einige der Entities fallen im Relationenmodell weg, da sie außer dem Primärschlüssel keine Attribute besitzen und keine relevante Information beinhalten. Meist wird dann der Name des Entities für eine Relation verwendet, die aus einer zugehörigen Relationship entsteht. Dazu gehören:

- "is-member-of" (Abb. 5.2) wird zur Relation `SphereT`
- "is-in-relation" (Abb. 5.3) wird zur Relation `Relationship`

- "belongs-to" (Abb. 5.3) wird zur Relation `Role`
- "has-competence" (Abb. 5.3) wird zur Relation `Competences`
- "has-skill" (Abb. 5.3) wird zur Relation `Skills`

5.4. Die Relationen

Das Relationenschema enthält die im folgenden detailliert aufgeführten Relationen. Es sind alle benutzten Relationen aufgeführt, nicht nur die, die aus dem ER–Diagramm abgeleitet sind.

5.4.1. Die Template–Relationen

Relation: `NodeT`

In dieser Relation werden alle grundlegenden Daten eines Knotens gespeichert.

nodeID Eindeutiger Identifikator für einen Knoten.

processID Identifiziert die Prozeßebene, die diesen Knoten realisiert.

type `= {activity, block, subprocess, toplevel}`
Gibt den Typ des Knoten an.

x, y, w, h Koordinaten für die grafische Darstellung des Knotens innerhalb des Monitors und des Workflow–Editors

icon Pfad bzw. Dateiname eines Icons, das einen Knoten symbolisieren soll. Der vollständige Pfadname wird mit Hilfe einer Environmentvariable auf folgende Art bestimmt: $SURRO_SPECPATH/icon/⟨Wert von Attribut icon⟩

iconText Beschriftung des Icons in der grafischen Darstellung, wird auch als Aktivitätenname verwendet.

startCondition Logischer Ausdruck, welcher die Auswertung der Kontrollverbindungen definiert. Innerhalb des Ausdrucks wird jedem Identifikator eines Kontrollkonnektors (connectorID) ein %–Zeichen vorangestellt.

event Ereignisausdruck, welches vor dem Starten der Aktivität eingetreten sein muß. Als Ereignisse können externe Ereignisse, absolute und relative Zeitereignisse, sowie zusammengesetzte Ereignisse spezifiziert werden. Es kann auch zwischen prozeßlokalen und prozeßglobalen Ereignissen unterschieden werden.

preCondition Logischer Ausdruck, welcher ausgewertet wird, nachdem das Ereignis (beschrieben in Attribut event) eingetreten ist.

exitCondition Logischer Ausdruck, welcher bei einem Block als Wiederholbedingung verwendet wird und bei einer Aktivität dazu benutzt wird, festzustellen, ob die Aktivität als beendet angesehen werden kann (eine Nachbedingung).

successCondition momentan nicht benutzt

description Ausführliche Beschreibung des Knotens und des Arbeitsschrittes, den der Bearbeiter ausführen soll.

inputContainerID Identifiziert den Container, welcher die Eingabedaten für diesen Knoten enthält.

outputContainerID Identifiziert den Container, welcher die Ausgabedaten dieses Knotens enthält.

failedCompNodeID Der hier angegebene Knoten wird gestartet, wenn der aktuelle Knoten erfolglos beendet wurde. Mit diesem Knoten soll der Arbeitsschritt des Knotens kompensiert werden.

successCompNodeID Dieser Kompensationsknoten wird dazu benutzt, wenn der erfolgreich ausgeführte Knoten kompensiert werden muß.

Relation: `NodeActivityT`

Die Relation stellt eine Spezialisierung der Relation `NodeT` dar. In dieser Relation werden die zusätzlichen Attribute eines Knotens gespeichert, wenn dieser eine Aktivität ist.

nodeID Eindeutiger Identifikator für einen Knoten (vgl. Relation `NodeT`).

programID Fremdschlüssel zu der Relation `Programs`. Beschreibt, welches Programm innerhalb dieser Aktivität zu starten ist.

humanResource Ausdruck zur Beschreibung der Auswahl eines Bearbeiters für diese Aktivität. Kann auch eine Rolle, etc. enthalten.

quantity Gibt an, wieviele Instanzen dieser Aktivität angelegt werden sollen. Momentan nicht benutzt.

priority Gibt die Priorität dieser Aktivität an. Je höher der Wert, umso höher ist die Priorität.

skill Auflistung (Tcl–Liste) der Fähigkeiten, die ein Bearbeiter haben muß, um diese Aktivität auszuführen.

competence Auflistung (Tcl–Liste) der Kompetenzen, die ein Bearbeiter haben muß, um diese Aktivität auszuführen.

responsibility Auflistung (Tcl–Liste) der Personen, die benachrichtigt werden sollen, wenn diese Aktivität nicht erfolgreich ausgeführt werden konnte.

clearUp Diese Funktion im Anwendungsprogramm wird aufgerufen, wenn eine Aktivität vorzeitig abgebrochen werden soll. Das Programm hat so die Gelegenheit, einen sicheren Zustand zu reichen, bevor es abgebrochen wird. Momentan nicht benutzt.

timeoutReady Zeitangabe, nach der ein Alarm aktiviert wird, wenn die Aktivität im Ready–Zustand ist und in dieser Zeit nicht durch den Benutzer aktiviert wurde (Angabe in Sekunden).

timeoutRunning Zeitangabe, nach der ein Alarm aktiviert wird, wenn die Aktivität im Running–Zustand ist und in dieser Zeit nicht durch den Benutzer beendet wurde (Angabe in Sekunden).

timeoutNodeID Ersatzaktivität, welche gestartet werden soll, wenn diese Aktivität einen Timeout erreicht hat.

timeoutMode Gibt an, wie das System reagieren soll, wenn diese Aktivität einen Timeout erreicht hat. Mögliche Modi sind: replace (Der normale Knoten wird abgebrochen und durch den Ersatzknoten ersetzt), onceAdditional (Der normale Knoten läuft weiter und der Ersatzknoten wird einmal zusätzlich gestartet, d. h. der Timeout wird nicht mehr gesetzt), manyAdditional (Nach den Timeout wird ein zusätzlicher Ersatzknoten gestartet), noMore (d. h. keine weiteren Knoten starten).

replaceNodeID Ersatzaktivität, welche gestartet werden soll, wenn diese Aktivität nicht erfolgreich terminieren konnte.

replaceMode Gibt an, wie das System reagieren soll, wenn diese Aktivität ersetzt werden soll. Mögliche Modi sind: replace, onceAdditional, manyAdditional, noMore.

Relation: `NodeBlockT`

Die Relation stellt eine Spezialisierung der Relation `NodeT` dar. In dieser Relation werden die zusätzlichen Attribute gespeichert, wenn dieser ein Block ist.

nodeID Eindeutiger Identifikator für einen Knoten.

defProcessID Dieser Block ist in dieser Prozeßebene definiert.

Relation: `NodeProcessT`

Sie stellt eine Spezialisierung der Relation `NodeT` dar. In dieser Relation werden die zusätzlichen Attribute gespeichert, wenn dieser ein Prozeß ist.

nodeID Eindeutiger Identifikator für einen Knoten.

defProcessID Dieser Prozeß ist in dieser Prozeßebene definiert.

processCompID Alle Kompensationsknoten werden auf dieser Prozeßebene gespeichert.

processReplaceID Alle Ersatzknoten werden auf dieser Prozeßebene gespeichert.

version Versionsnummer des Prozesses. Wird benötigt bei Ad–hoc–Modifikationen.

portfolio Verweis auf ein Verzeichnis mit Dokumenten, die zum Vorgang gehören.

Relation: `Programs`

Diese Relation stellt einen Pool von Programmen zur Verfügung, die in den einzelnen Aktivitäten aufgerufen werden können.

programID Eindeutiger Identifikator eines Programms.

platform Plattform auf der das Programm ausgeführt werden kann (z.B. java, tcl, unix, win, linux, sunOS, AIX, OS2, HPUX, obj).

command Kommandozeile, um das Programm aufzurufen.

programName Name des Programms. Wird vom Programmpool zur Identifikation der Anwendung benutzt, die gegebenfalls installiert wird.

Relation: `ContainerT`

Die Relation enthält alle Variablen der Input– und Outputcontainer.

variableID Eindeutiger Identifikator für eine Variable. Eine Variable ist einem Container eindeutig zugeordnet.

containerID Eindeutiger Identifikator eines Containers. Ein Container enthält eine Menge von Variablen.

processID Dieser Prozeßebene ist der Container zugeordnet.

containerType Gibt an, um welchen Typ von Container es sich handelt (xput, source, sink). xput steht für Input– oder Outputcontainer.

name Name der Containervariablen.

type Datentyp der Containervariablen.

value Wert der Containervariablen in einer Zeichenfolge.

Relation: ControlConnectorT

Die Relation enthält alle Kontrollkonnektoren, die zwischen den Aktivitäten bestehen.

connectorID Eindeutiger Identifikator eines Kontrollkonnektors.

processID Der Konnektor ist dieser Prozeßebene zugeordnet.

condition Bedingung, die erfüllt sein muß, daß diese Verbindung zu "true" evaluiert wird.

fromNodeID Startknoten, von dem der Kontrollkonnektor ausgeht.

toNodeID Endeknoten, auf den der Kontrollkonnektor zeigt.

Relation: DataConnectorT

Die Relation enthält alle Datenkonnektoren, die zwischen den Aktivitäten bestehen.

connectorID Eindeutiger Identifikator eines Datenkonnektores.

processID Der Konnektor ist dieser Prozeßebene zugeordnet.

fromContainerID Aus diesem Container werden die Variablen herauskopiert.

toContainerID In diesen Container werden die Variablen hineinkopiert.

Relation: MappingT

Die Relation beschreibt die Umsetzung der Variablen von einem Container in einen anderen.

connectorID Eindeutiger Identifikator eines Datenkonnektores.

fromVariableID Datenquelle für die Umsetzung.

toVariableID Datensenke für die Umsetzung.

function Diese Funktion wird beim Umkopieren auf die Variable angewendet.

Relation: SphereT

Die Relation beschreibt eine Sphäre

sphereID Eindeutiger Identifikator für eine Sphäre.

processID Dieser Prozeßebene ist die Sphäre zugeordnet.

type Gibt den Type der Sphäre an (TaS oder CS) (d.h. transaction sphere oder compensation sphere).

compensationNodeID Die gesamte Sphäre kann mit diesem Kompensationsknoten kompensiert werden.

restartMode Gibt an, was nach einem Rücksetzen der Sphäre gemacht werden soll (retry, undo). Momantan nicht benutzt.

Relation: `SphereMemberT`

Die Relation beschreibt, welcher Knoten an welcher Sphäre teilnimmt.

sphereID Eindeutiger Identifikator für eine Sphäre.

nodeID Eindeutiger Identifikator für einen Knoten.

5.4.2. Die Organisations–Relationen

Mit den nachfolgenden Relationen kann die Aufbauorganisation eines Unternehmens oder Behörde beschrieben werden. Das Organisationsmodul verwendet die folgenden Relationen zur Bestimmung des Bearbeiters einer Aktivität.

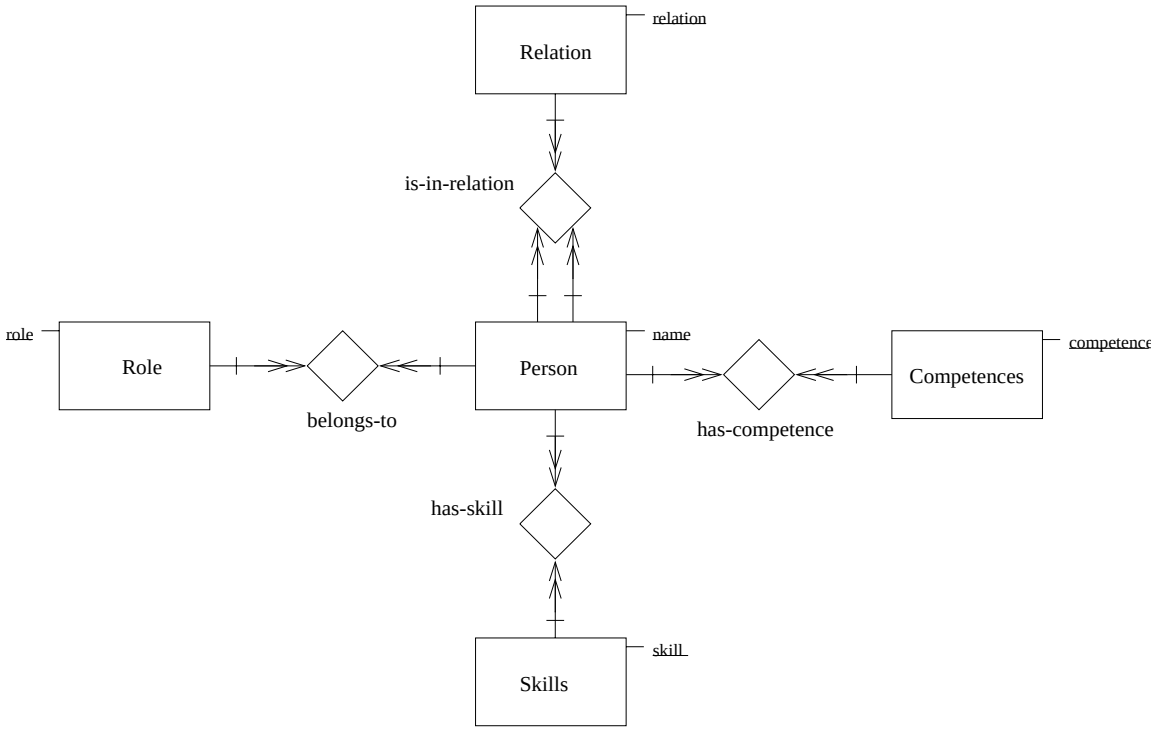

Abbildung 5.3.: Das ER–Diagramm zur Beschreibung der Aufbauorganisation

In Abb. 5.3 bezieht das Entity "Relation" seine Existenzberechtigung aus der Tatsache, daß zwischen denselben zwei Personen mehrere verschiedene Beziehungen

bestehen können. Ansonsten wäre dieses Entity sinnvollerweise eine (wie der Name schon sagt) Relationship.

In der Relation `NodeActivity(T)` wird in einem "humanResource"–Ausdruck bestimmt, welche Person die Aktivität ausführen soll. Der Ausdruck folgt der hier in EBNF angegebenen Syntax:

$$humanResource ::= (person \mid role \mid relationship\ (person \mid role)\ )$$

Eine konkrete Person kann also direkt angegeben werden (z. B. humanResource = 'Schreyjak'). Es kann eine Rolle spezifiziert werden (z. B. humanResource = 'Mitarbeiter'). Es kann auch auf eine Beziehung zurückgegriffen werden und so z. B. der Chef der Mitarbeiter angegeben werden (humanResource = 'is chef of Mitarbeiter'). Eine Rolle ist als eine Menge von Personen definiert. Eine Beziehung drückt ein beliebiges Verhältnis zwischen Rollen oder Personen aus. Jede Person kann mehrere Fähigkeiten oder Kompetenzen besitzen, die frei gewählt werden können.

Relation: `Person`

Diese Relation beschreibt alle Attribute, die einer Person direkt zugeordnet werden können.

name eindeutiger Verwaltungsname einer am Workflowsystem teilnehmenden Person

icon Filename, in dem ein Bild der Person gespeichert ist.

organisation Name der Organisation, in der diese Person angestellt ist.

firstname Vorname der Person

lastname Nachname der Person

occupation Beruf der Person

substitute Mit diesem Ausdruck wird die Vertreterregelung angegeben: Wenn der Ausdruck leer ist, gibt es keine Vertreterregel für diese Person. Steht der Name eine anderen Person darin, werden die zu bearbeitenden Aktivitäten an diese Person weitergeleitet. Wenn eine Rolle angegeben ist, wird die Rolle in eine konkrete Person aufgelöst. Steht im Ausdruck eine Beziehung zu einer Rolle oder einer Person, so wird diese Beziehung verfolgt und dann entsprechend eine Auslösung vorgenommen.

Relation: `RoleMapping`

In dieser Relation wird beschrieben, welche Personen an welchen Rollen teilnehmen.

role Bezeichnung der Rolle.

name Name der Person, die dieser Rolle wahrnimmt.

Relation: `Competences`

In dieser Relation wird beschrieben, welche Kompetenzen die Personen besitzen.

person Name der Person, deren Kompetenzen beschrieben werden.

competence Die Kompetenz, die diese Person hat.

Relation: `Skills`

In dieser Relation wird beschrieben, welche Fähigkeiten die Personen besitzen.

person Name der Person, deren Fähigkeit beschrieben wird.

skill Die Fähigkeit, die diese Person hat.

Relation: `Relationship`

In dieser Relation wird beschrieben, welche Beziehungen der Personen bzw. Rollen untereinander haben. Um keine explizite Unterscheidung zwischen Person und Rolle machen zu müssen, wird von Ressourcen gesprochen.

resource1 Die erste Ressource innerhalb einer Beziehung.

relation Die Beziehung zwischen den beiden Ressourcen.

resource2 Die zweite Ressource innerhalb einer Beziehung.

Relation: `Participation`

In dieser Relation wird beschrieben, welche Personen an welcher Warteschlage teilnehmen. Bei einer Warteschlage muß der Bearbeiter seine Aktivitäten selber abholen. Eine Aktivität kann nur von einer Person bearbeitet werden. Momentan nicht benutzt.

person Name einer Person, die an der Warteschlage teilnimmt.

queue Name der Warteschlagen, an der die Person teilnimmt.

5.4.3. Die Verwaltungs–Relationen

Relation: `Events`

In dieser Relation werden zu erwartende (time ist leer) und aufgetretene Events (time wurde gefüllt) gespeichert.

eventID Eindeutiger Identifikator für einen Event.

eventName Textuelle Identifikation eines Events.

type = `{local, global}` Der Typ eines Ereignisses gibt an, ob das Ereignis für alle Workflows gibt (global) oder ob es nur innerhalb eines Workflows Gültigkeit besitzt.

time Beschreibung des Zeitpunktes, wann der Event aufgetreten ist. Wenn dieses Attribut leer ist, ist das Ereignis noch nicht aufgetreten.

origin Gibt an, wer (z. B. eine Aktivität oder eine Person) das Auftreten des Ereignisses gemeldet hat.

OSProcess Dieses Attribut wird momentan für die Speicherung der Workflow–Instanz–ID benutzt, in dessen Rahmen das Ereignis aufgetreten ist.

Relation: `SequenceGlobal`

In dieser Relation werden alle Zähler, die eindeutige Identifikatoren liefern, persistent gespeichert.

processID Speichert die eindeutigen Prozeßebenen–Nummern.

wfID Speichert die eindeutigen Workflow–Instanz–Nummern.

eventID Speichert die eindeutigen Event–Nummern.

sphereID Speichert die eindeutigen Sphären–Nummern.

containerID Speichert die eindeutigen Container–Nummern.

variableID Speichert die eindeutigen Variablen–Nummern.

connectorID Speichert die eindeutigen Konnektor–Nummern.

Relation: `SequenceLocal`

In dieser Relation werden alle Zähler gespeichert, die nur innerhalb eines Workflows eindeutig sind.

wfInstanceID Gibt an, für welche Workflow–Instanz diese Zähler gültig sind.

blockInstanceID Zähler für die Blockinstanzen innerhalb eines Workflows.

sphereInstanceID Zähler für die Sphäreninstanzen innerhalb eines Workflows.

5.4.4. Die Instanz–Relationen

Die folgenden Relationen sind von den Template–Relationen abgeleitet. Es werden daher nur die zusätzlichen Attribute beschrieben, die in den Template–Relationen nicht zu finden sind.

Relation: `Node`

Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

Relation: `NodeActivity`

Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

state `={initial, pending, ready, running, successful, failed, compensated, terminated}` Namen der Zustände einer Aktivität

Relation: `NodeBlock`

Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

state `={initial, pending, active, finished, terminated}`
Namen der Zustände eines Blocks

Relation: `NodeProcess`

Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

starter Person, die diese Prozeßinstanz gestartet hat (nur bei Toplevel–Prozesse).

state `={initial, pending, running, finished, terminated}`
Namen der Zustände eines Prozesses

Relation: `Container`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

Relation: `ControlConnector`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

state `={false, true, undefined, locked}`
Namen der Zustände eines Kontrollkonnektors

Relation: `DataConnector`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

Relation: `Mapping`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

Relation: `Sphere`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

state `={initial, active, backout, finished, committed}`
Namen der Zustände einer Sphäre

Relation: `SphereMember`
Siehe auch die Attribute in der Template–Relation

wfInstanceID Eindeutiger Identifikator einer Workflow–Instanz

blockInstanceID Eindeutiger Identifikator einer Prozeßebeneninstanz

5.5. Randbedingungen im Datenmodell

- Jeder Outputcontainer sollte die Stringvariable complete_info und die Integervariable complete_state besitzen. In complete_info kann ein Ausgabetext (z. B. eine Fehlerbeschreibung) geschrieben werden kann und in complete_state sollte ein Rückgabewert geschrieben werden.

- Der Inputcontainer von Ersatzaktivitäten und Kompensationsaktivitäten ist aus dem Inputcontainer und dem Outputcontainer der normalen Aktivität zusammengesetzt. Die Container von normalen Aktivitäten und Ersatzaktivitäten, bzw. Kompensationsaktivitäten müssen daher denselben Aufbau besitzen, da hier kein Mapping der Containervariablen stattfindet, sondern ein einfaches Kopieren der Containerinhalte. Derselbe Mechanismus wird bei den Containern eines Blocks und den Source– und Sinkcontainern im Block benutzt.

  Der Outputcontainer der Ersatz– und Kompensationsaktivitäten darf nur aus den Rückgabewerten complete_info und complete_state bestehen. Es dürfen keine Datenkonnektoren von diesen Variablen wegführen.

6. Die Funktionsweise von Surro

6.1. Die Workflow–Engine

6.1.1. Der strukturelle Aufbau der Workflow–Engine

Nach dem Start der Workflow–Engine werden alle notwendigen Initialisierungen vorgenommen. Falls der Neustart nach einem Systemabsturz stattfindet, werden eventuell notwendig gewordene Maßnahmen für ein Recovery ausgeführt. Danach wartet die Engine auf Nachrichten, die in einer Nachrichtenwarteschlange gespeichert werden. Die Nachrichten heißen interne Ereignisse. Mit dem Entnehmen eines internen Ereignisses wird in allgemeinen eine Transaktion gestartet. Alle Änderungen des Workflowzustands, der in der Datenbank gespeichert ist, werden nun im Rahmen dieser Transaktion ausgeführt. Während der Verarbeitung eines internen Ereignisses kann die Engine neue interne Ereignisse erzeugen, die ebenso in der Nachrichtenwarteschlange für die spätere Bearbeitung gespeichert werden. Nach dem Beenden der Verarbeitung eines internen Ereignisses wird die Transaktion beendet. Der Workflow hat damit einen neuen Zustand. Der Zustandsübergang wurde durch eine Transaktion geschützt. Das nächste Ereignis in der Warteschlange wird nun gelesen und verarbeitet.

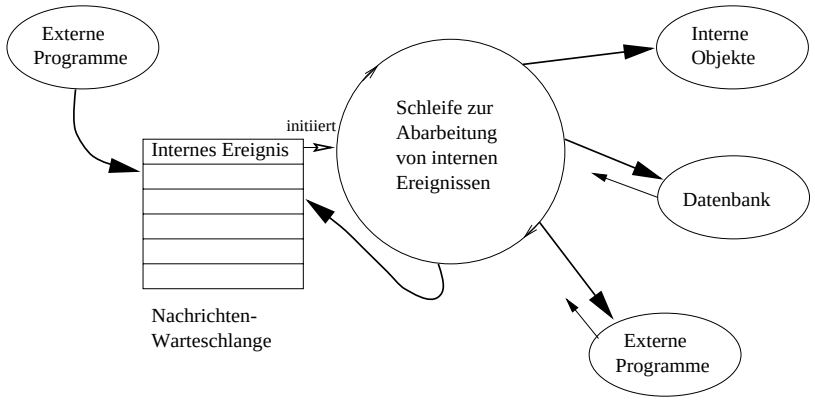

Abbildung 6.1.: Eine Abarbeitungsschleife als prinzipielle Arbeitsweise der Workflow–Engine

Aufträge externer Programme an die Engine werden durch das Einfügen eines

internen Ereignisses in die Warteschlange abgesetzt. Bei der Verarbeitung eines internen Ereignisses durch die Engine in Form einer Schleife kann Kommunikation zu anderen Programmen notwendig werden. Die Kommunikation findet hier synchron statt, d. h. die Engine erwartet eine sofortige Antwort.

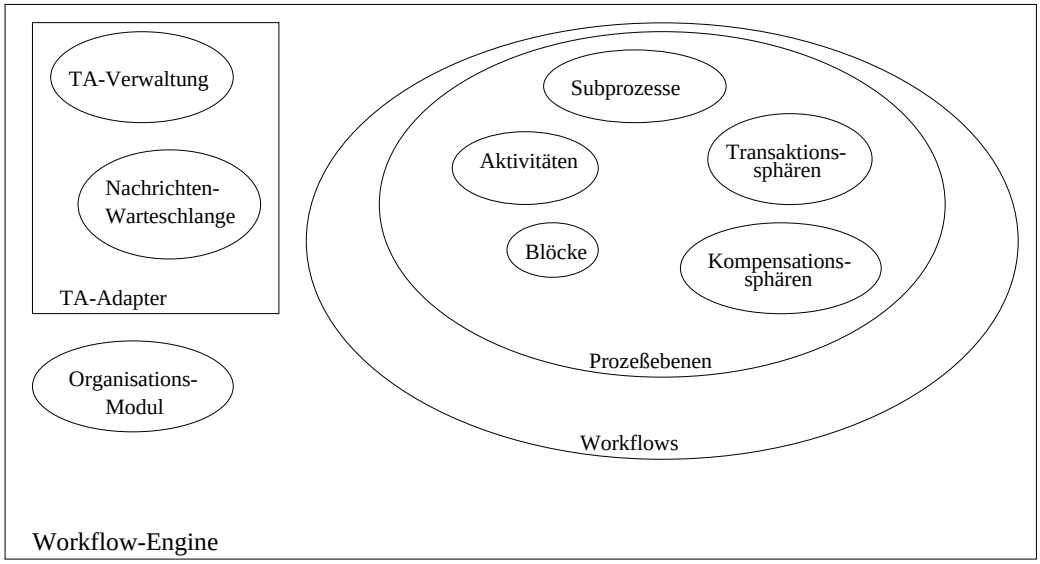

Abbildung 6.2.: Der Aufbau der Engine nach einzelnen Objekten strukturiert

In Abbildung 6.2 ist der innere Aufbau der Workflow–Engine nach Objekten strukturiert dargestellt. Ein Workflow besteht aus mehreren Prozeßebenen (siehe 49). Eine Prozeßebene beinhaltet wiederum die Objekte Aktivität, Block und Subprozeß. Daneben gibt es die Transaktions– und Kompensations–Sphären–Objekt. Die Objekte für die Transaktion–Verwaltung und die Nachrichtenwarteschlange sind in den OS/2–Prozeß TA–Adapter ausgelagert, gehören aber prinzipiell zur Engine. Das Organisationsmodul kann auch als eigenständiges Programm implementiert werden, ist hier aber als internes Engine–Objekt aufgeführt.

Die Objekte Aktivitäten, Blöcke, Subprozeß, Transaktions–Sphäre und Kompensations–Sphären sind als Zustandsautomaten realisiert. Der Zustände der Objektinstanzen werden in der Datenbank gespeichert. Die einzelnen Objekte und ihre Implementierungen werden in den folgenden Abschnitten detaillierter erläutert.

6.1.2. Die Nachrichtenwarteschlange

Die Engine besitzt eine Abarbeitungsschleife, in der sie auf das Auftreten von internen Ereignissen wartet und diese dann abarbeitet. Die Ereignisse werden in einer FIFO–Nachrichtenwarteschlange gespeichert. Diese Warteschlange ist als Resource–Manager in Form eines DSOM–Objektes der Klasse `TAQueue` implementiert (transaktionale Queue). Die Aufgabe der transaktionalen Queue ist es, Nachrichten bzw. Ereignisse sicher zu speichern.

Die Queue besitzt als Schnittstelle die Operationen `enqueue` zum Eintragen eines Elementes an das Ende der Queue, `dequeue` zum Entfernen des nächsten Elementes, `peek` zum nicht–zerstörenden Lesen des nächsten Elements, `count_entries` zur Ermittlung zur Anzahl der vorhandenen Elemente und `get_ta_handle` zum Ermitteln des Transaktionskontextes des nächsten Elements.

Nachrichten müssen innerhalb eines gültigen Transaktionskontexts in die Queue geschrieben und aus ihr ausgelesen werden. Ein Element, das in einer aktiven Transaktion in die Queue gestellt worden ist, kann auch nur im Kontext genau dieser Transaktion wieder ausgelesen werden. Damit die Engine beim Auslesen den richtigen Kontext verwendet, ist sie in der Lage mittels `get_ta_handle` den Kontext zu ermitteln. Ist das Element in einer Transaktion eingetragen worden, die mittlerweile erfolgreich beendet worden ist (Das Element ist dann „bestätigt“, engl. committed), so kann das Element in einem (beliebigen) Kontext ausgelesen werden. Wird eine Transaktion zurückgesetzt, so werden alle Elemente der Queue, die in diesem Kontext gespeichert worden sind, wieder entfernt. Analog werden bestätigte Elemente, die in einem anderen Kontext ausgelesen worden sind, bei dessen Rollback wieder in die Queue zurückgeschrieben.

Die Arbeitsweise der Warteschlange

In Abbildung 6.3 wird dieser Vorgang und der Gebrauch der transaktionalen Warteschlange durch die Workflow–Engine verdeutlicht.

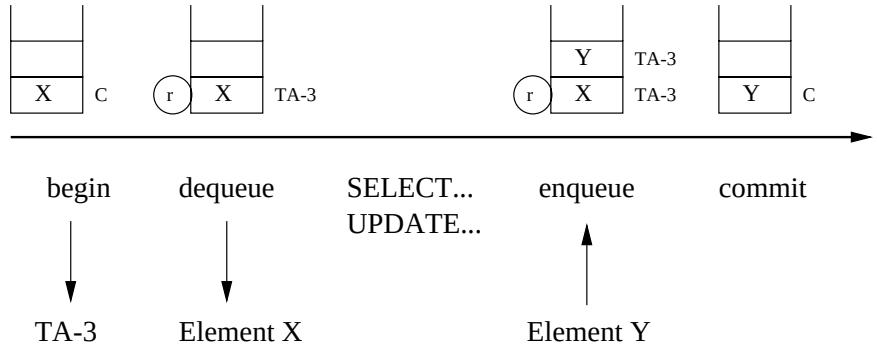

Abbildung 6.3.: Der Gebrauch der transaktionalen Warteschlange durch die Engine

Die Engine beginnt eine neue Transaktion und erhält im Beispiel das Handle TA-3, das eine Referenz auf den OTS–Transaktionskontext darstellt. Alle weiteren Operationen auf Resource–Managern werden nun in diesem Kontext durchgeführt. In der Queue befindet sich das Element X im bestätigten Zustand (angedeutet durch die Markierung c für “committed”). Die Engine liest das Element aus, das dadurch als innerhalb von TA-3 gelöscht markiert wird (Markierung r steht für “removed”). Das Element wird nicht entfernt, da es bei einem Rollback der Transaktion restauriert werden muß.

Das Element X wird nun von der Engine verarbeitet und löst dabei diverse Operationen aus (UPDATE, SELECT), beispielsweise um den Zustand einer Aktivität auf der Workflow–Datenbank zu aktualisieren. Nach Beendigung dieser Operationen wird typischerweise wiederum eine Nachricht (Y) auf die Queue geschrieben, die weitere Folgeoperationen auslösen soll. Danach wird die Transaktion beendet. Erst dadurch wird das Element X endgültig von der Queue entfernt. Das Element Y erhält den Zustand "committed". Es kann auch vorkommen, daß keine weitere Nachricht Y in die Queue eingetragen wird. In diesem Fall arbeitet die Engine erst dann weiter, wenn durch ein externes Programm (z. B. durch den Aktivitäten–Manager) eine Nachricht (ein externes Ereignis) in die Warteschlange eingereiht wird.

Durch das transaktionale Verhalten der persistenten Warteschlange wird erreicht, daß im Fehlerfall, also bei einem Transaktionsabbruch, jeweils die korrekten Nachrichten beim Wiederanlauf in der Queue stehen und somit gleich eine Wiederholung der abgebrochenen Vorgänge durchgeführt werden kann.

Liste der definierten internen Ereignissen

Als *interne Ereignisse* werden Nachrichten bezeichnet, die zur Steuerung der Workflow–Engine in die Nachrichtenwarteschlange geschrieben werden. Interne Ereignisse sind häufig Auslöser von Zustandsübergängen bei Aktivitäten, Blöcken und Sphären. Folgende interne Ereignisse sind definiert:

startWorkflow	Ein Workflow soll gestartet werden.
actorStart	Der Bearbeiter zeigt an, daß er eine Aktivität gestartet hat.
actorSuccessful	Der Bearbeiter zeigt an, daß er eine Aktivität erfolgreich beendet hat.
actorFailed	Der Bearbeiter zeigt an, daß er eine Aktivität erfolglos beendet hat.
actorRefused	Der Bearbeiter zeigt an, daß er eine Aktivität nicht bearbeiten will.
actorOmitted	Der Bearbeiter zeigt an, daß er eine optionale Aktivität nicht bearbeiten will und sie deshalb ausläßt.
backout	Eine Kompensations–Sphäre soll wieder in den Anfangszustand versetzt werden. Dazu müssen alle bearbeiteten Aktivitäten kompensiert werden.
timeoutReady	Das Zeitlimit einer Aktivität im Zustand `ready` ist überschritten worden.

timeoutRunning	Das Zeitlimit einer Aktivität im Zustand `running` ist überschritten worden.
startReplaceNode	Ein Ersatzknoten soll gestartet werden.
startCompensationNode	Ein Kompensationsknoten soll gestartet werden.
restartNode	Ein Knoten soll erneut gestartet werden (nach einem backout).
eventOccured	Das zum Knoten gehörige Ereignis ist aufgetreten.
suE_*	Die Workflow–Engine bietet verschiedene Prozeduraufrufe an, die mit diesen Nachrichten von externen Programmen aufgerufen werden können.

6.1.3. Das Transaktionskontext–Verwaltungsobjekt

Das Transaktionskontext–Verwaltungsobjekt ist in das OS/2 Programm "TA–Adapter" ausgelagert. Der TA–Adapter fungiert als Client für alle transaktionale Vorgänge. Er tritt aus Sicht des OTS als Initiator für sämtliche transaktionalen Operationen auf. Logisch gesehen müßte die Engine diese Aufgabe übernehmen, kann dies aber wegen der Nichtverfügbarkeit von SOM 3.0 unter UNIX nicht übernehmen. Daher wird der zusätzliche Prozeß TA–Adapter als „OS/2–Verlängerung" der Engine und eine Schnittstelle zwischen den Programmen geschaffen. Diese bietet Funktionen zum Starten (`begin`), Beenden (`commit` und `rollback`) von Transaktionen und zum expliziten Setzen des Transaktionskontextes (`set_context`). Zur Zeit werden geschachtelte Transaktionen noch nicht unterstützt. Jede Transaktion ist eine Top–level–Transaktion. Über das Interface erhält die Engine zu jeder neuen Transaktion ein eindeutiges Handle, das als Parameter für `set_context` benötigt wird. Nach Beginn einer Transaktion bzw. einem expliziten Kontextwechsel werden alle transaktionalen Operationen implizit im Rahmen dieser Transaktion durchgeführt. Es ist nicht notwendig, bei jeder Operation den Kontext als Parameter mitzuliefern.

Der TA–Adapter muß somit eine Abbildungsfunktion von Transaktions–Handle zum realen Transaktionskontext bereitstellen. Dies wird mittels eines eigenen DSOM–Objekts der Klasse "TAContext" realisiert, damit auch andere Prozesse (z. B. der PEC) auf die Kontextinformation zugreifen können.

6.1.4. Das Workflow Objekt

Das Workflow Objekt beinhaltet in der Hauptsache alle bereits instanziierten Prozeßebenen. Es werden immer nur die Prozeßebenen instanziiert, in die der Kontrollfluß eingetreten ist. Eine Workflow–Instanz–ID identifiziert einen Workflow. Die Engine kann mehrere Workflows parallel bearbeiten.

6.1.5. Das Prozeßebenen Objekt

Eine Prozeßebene beinhaltet Aktivitäten, Blöcke, Subprozesse und die Sphären, die auf derselben Ebene liegen. Alle Objekte einer Prozeßebene werden mit dieser zusammen instanziiert. Blöcke und Subprozesse sind Stellvertreter für weitere, tieferliegende Prozeßebenen.

6.1.6. Das Aktivitäten Objekt

Eine Aktivität besitzt verschiedene Zustände. In Abbildung 6.4 ist das zugehörige Zustandsdiagramm einer Aktivität dargestellt.

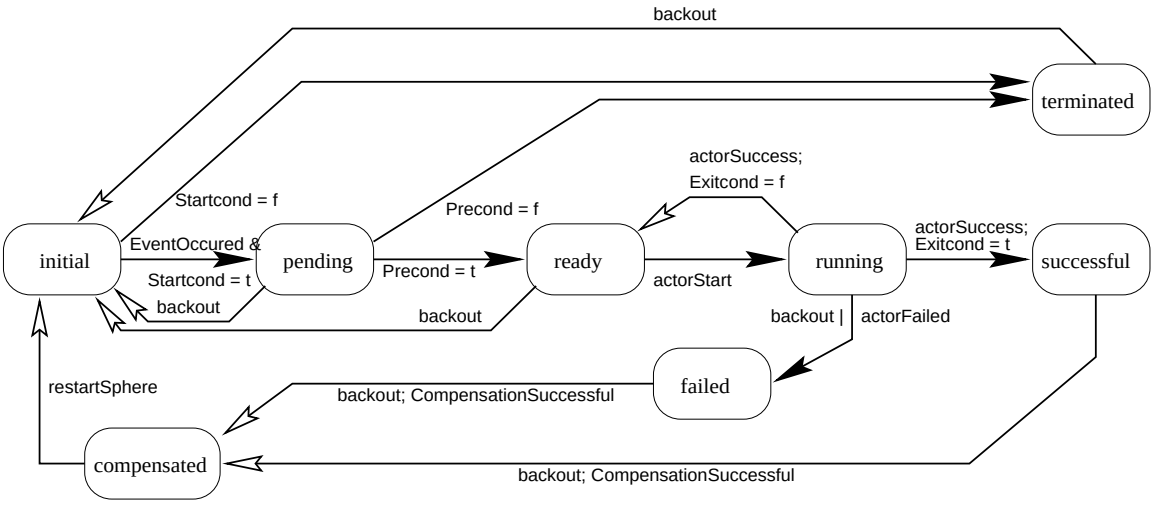

Abbildung 6.4.: Das Zustandsdiagramm einer Aktivität

Nach der Instanziierung befindet sich eine Aktivität im Zustand INITIAL. Wenn die Aktivität eine Startaktivität ist, oder wenn der Kontrollfluß die Aktivität erreicht und alle eingehenden Kontrollflußkonnektoren entweder zu wahr oder falsch evaluiert wurden, dann wird die Startbedingung der Aktivität ausgewertet. Falls sie positiv ausfällt, wechselt die Aktivität in den Zustand PENDING. Hierin wartet sie auf das Auftreten eines spezifizierten Ereignisses. Falls kein Ereignis spezifiziert oder falls das Ereignis eingetreten ist, wechselt die Aktivität in den Zustand READY. Dabei findet eine Zuteilung der Aktivität zu einer konkreten Person statt. Falls eine Rolle als Akteur angegeben ist, muß eine Rollenauflösung durch das Organisationsmodul stattfinden. Die Aktivität erscheint auf der Arbeitsliste des ausgewählten Bearbeiters. Der Bearbeiter hat die Wahl, diese Aktivität zu bearbeiten. Im Bearbeitungsfall geht die Aktivität in den Zustand RUNNING über. Eine Aktivität kann entweder erfolgreich oder erfolglos bearbeitet werden. Wenn der Bearbeiter angegeben hat, daß die Aktivität erfolgreich bearbeitet wurde, dann prüft die Workflow–Engine, ob sie dies mit Hilfe der Nachbedingung verifizieren kann. Falls die Nachbedingung

nicht verifiziert werden kann, kommt die Aktivität erneut im Zustand READY auf die Arbeitsliste desselben Bearbeiters.

In den Zustand TERMINATED kommt die Aktivität, wenn die Startbedingung zu falsch evaluiert wird (Dies kann insbesondere bei der Dead–Path–Elimination stattfinden, s.u.). Wenn nach dem Eintreten des spezifizierten Ereignisses die Vorbedingung nicht zutrifft, wird ebenfalls der Zustand TERMINATED erreicht.

Wenn sich die Aktivität in einer Kompensations–Sphäre befindet, dann kann ein „backout“ ausgelöst werden, d. h. die Ausgangssituation zu Beginn der Sphäre muß wiedergestellt werden. Falls die Aktivität noch nicht bearbeitet wurde, kann sie einfach wieder in den Ausgangszustand versetzt werden. Aber falls sie sich noch in Arbeit befindet oder schon bearbeitet worden ist, dann muß sie kompensiert werden. Falls die Kompensation erfolgreich war, tritt die Aktivität in den Zustand kompensiert über. Von dort geht es erst dann in den Zustand INITIAL weiter, wenn alle Aktivitäten der Sphäre erfolgreich kompensiert wurden.

6.1.7. Das Block Objekt

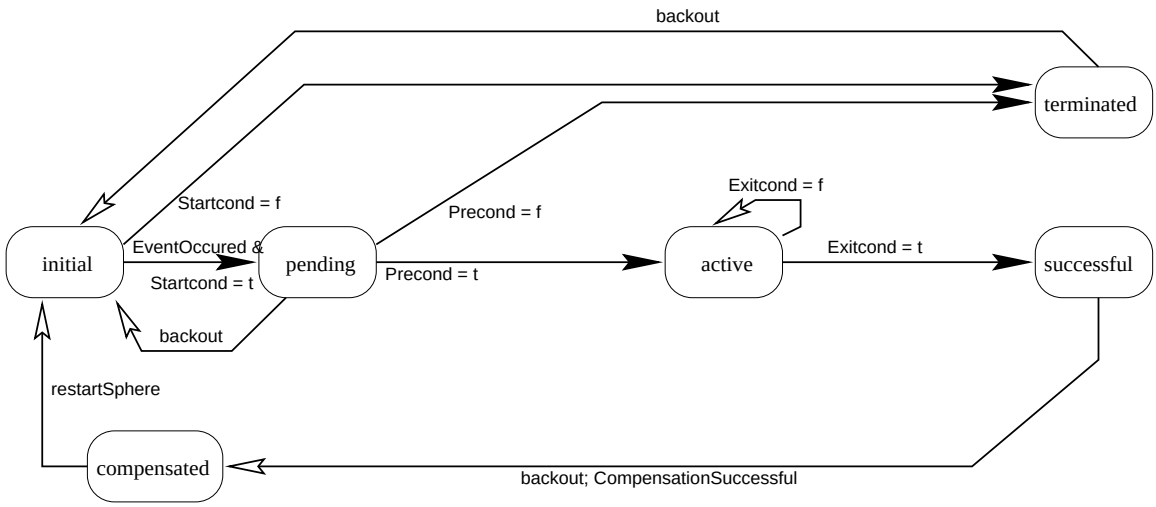

Abbildung 6.5.: Das Zustandsdiagramm eines Blocks

Das Zustandsdiagramm eines Blocks ähnelt dem der Aktivität, mit dem Unterschied, daß Blöcke nicht an Bearbeiter verteilt werden und daher der Zustand READY nicht existiert. Auch kann ein Block nicht erfolglos beendet werden. Erfolgreich ist er beendet, wenn alle Knoten innerhalb des Blocks in einem Endzustand sind. Die Exitcondition wird als Wiederholbedingung für den gesamten Block verwendet, damit wird eine Schleifenkonstruktion ermöglicht.

6.1.8. Das Subprozeß Objekt

Das Subprozeß Objekt ist nicht implementiert worden (Stand Ende 1996).

6.1.9. Das Transaktions–Sphären Objekt

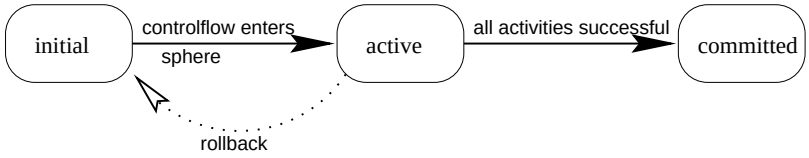

Abbildung 6.6.: Das Zustandsdiagramm einer Transaktions–Sphäre

Eine Sphäre wird zusammen mit den Aktivitäten, Blöcken usw. . einer Prozeßebene instanziiert und befindet sich dann im Zustand INITIAL. Sobald der Kontrollfluß die Sphäre betritt, wechselt die Sphäre den Zustand nach ACTIVE. Wenn alle Aktivitäten erfolgreich beendet worden sind, initiiert die Workflow–Engine mit einem commit das 2–Phasen–Commit–Protokoll, das bei positivem Ausgang dazu führt, daß die Sphäre in den Zustand COMMITTED übergeht. Falls das 2PC–Protokoll zu einer negativen Entscheidung kommt, wird ein rollback ausgeführt und der Zustand wechselt implizit durch die Ausführung des Rollbacks in den INITIAL–Zustand über. Falls eine Aktivität einen erfolglosen Abschluß meldet, initiiert die Workflow–Engine ein rollback und beendet so vorzeitig die Bearbeitung der Sphäre.

6.1.10. Das Kompensations–Sphären Objekt

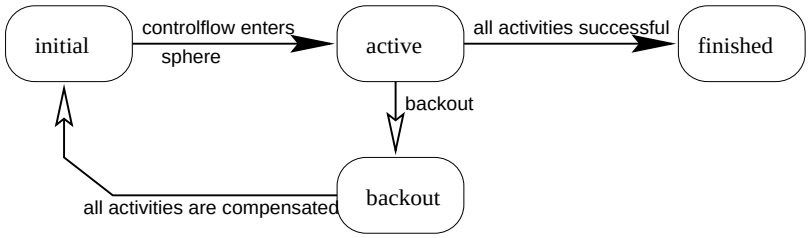

Abbildung 6.7.: Das Zustandsdiagramm einer Kompensations–Sphäre

Das Zustandsdiagramm der Kompensations–Sphäre unterscheidet sich von dem Zustandsdiagramm der Transaktions–Sphäre durch den zusätzlichen Zustand BACKOUT. Wenn eine Aktivität einen erfolglosen Abschluß meldet, wird ein backout ausgelöst und die Sphäre wechselt in den entsprechenden Zustand. Dieses backout–Signal wird an alle Knoten in der Sphäre weitergereicht, die nicht im Anfangszustand sind. Wenn dann alle Knoten entweder im Anfangszustand oder erfolgreich kompensiert sind, kann die Sphäre wieder in den INITIAL Zustand wechseln und erneut bearbeitet werden.

6.1.11. Das Organisationsmodul

Dieser Teil des Workflowsystem hat die Aufgabe, im Auftrag der Workflow–Engine einen Bearbeiter für eine Aktivität zu finden. In der Workflow–Spezifikation muß zu jeder Aktivität ein Ausdruck angegeben werden, über den das Organisationsmodul einen Bearbeiter aus der spezifizierten Aufbauorganisation des Unternehmens oder der Behörde heraussucht. Als Grundlage dient dazu das auf Seite 57 beschriebene Datenmodell der Aufbauorganisation.

In der Organisationsdatenbank (bzw. in den Organisations–Relationen der Workflow–Datenbank) müssen alle Teilnehmer am Workflowsystem, die *Bearbeiter*, aufgeführt sein. Zu jeder Person können eine Menge von Fähigkeiten und Kompetenzen angegeben werden. Die Fähigkeiten und Kompetenzen sind nicht vordefiniert und können während der Modellierung der Aufbauorganisation frei gewählt werden. Fähigkeiten und Kompetenzen können als Auswahlkriterium herangezogen werden. So kann z. B. eine Person durch eine andere Person vertreten werden, wenn sie dieselben Fähigkeiten besitzt, die zur Bearbeitung einer Aktivität nötig sind.

Eine Gruppe von Personen kann zu einer *Rolle* zusammengefaßt werden. Als Teilnehmer einer Rolle sind die Personen beliebig austauschbar, d. h. eine Teilnehmer einer Rolle kann durch einen anderen Teilnehmer an derselben Rolle ersetzt werden. Sie besitzen damit implizit dieselben Fähigkeiten und Kompetenzen. Das Organisationsmodul hat die Aufgabe, diese Auswahl einer Person über die Rolle nach bestimmten Algorithmen vorzunehmen. In der momentan implementierten Version wird der erste im Workflowsystem eingeloggte Bearbeiter ausgewählt. Weitere Auswahlalgorithmen, wie eine abwechselnde Auswahl oder die Auswahl des Bearbeiters mit der geringsten Arbeitslast, sind relativ einfach implementierbar.

Zwischen Rollen und Personen können frei bestimmbare *Verhältnisse* spezifiziert werden. Damit kann z. B. die hierarchische Struktur des Unternehmens abgebildet werden. Durch die freie Wahl der Verhältnisse können auch mehrere Hierarchiearten realisiert werden.

Das Organisationsmodul führt eine eigene Protokolldatei, mit der das Modul auf ein früher getroffene Auswahl zurückgreifen kann. Somit wird es möglich, eine Aktivität von demselben Bearbeiter wie eine andere Aktivität erledigen zu lassen. Zudem kommuniziert das Organisationsmodul mit dem Workflow–Session–Manager, um auf die Daten über die eingeloggten Bearbeiter zugreifen zu können.

6.1.12. Das Kommunikationsprotokoll

Zur Kommunikation über Sockets wird ein Nachrichtenformat verwendet, das zur Übertragung strukturierter Daten geeignet ist. Ein einzelnes Nachrichtenatom hat die Grundstruktur:

```
<type>:<length>:<content>
```

`<type>` ist der Typ der Nachricht, `<length>` gibt die dezimal ASCII–codierte Länge des folgenden `<content>`–Feldes an. `<content>` beinhaltet die Nutzinformation, die

aber wiederum aus gleich strukturierten Nachrichten aufgebaut sein kann. Aus dem Typ ist erkenntlich, ob die Nutzinformation aus einem String oder weiter strukturierten Nachrichtenatomen besteht. Beispiel:

```
struct1:19:s:5:hellos:6:World!
|          |_______||________|
|_____________________________|
```

ist eine Nachricht vom Typ "struct1", die die zwei Strings (Typ „s“) "hello" und "World!" enthält.

6.2. Interne Abarbeitung eines Workflows

Der Aktivitäten–Manager fügt das interne Ereignis "startWorkflow" in die Nachrichtenwarteschlange ein, damit die Engine einen Workflow startet. Die Engine liest die Nachricht und erzeugt eine neue Transaktion, in der dann der Workflow instanziiert wird. Es wird dabei nur die oberste Prozeßebene mit allen Aktivitäten, Blöcken und Sphären instanziiert. Anschließend werden alle Startaktivitäten ermittelt und die Startnachrichten in die Nachrichtenwarteschlange gestellt. Startaktivitäten sind alle Aktivitäten ohne eingehenden Kontrollflußkonnektoren. Die Instanziierungs–Transaktion wird beendet. Als nächstes werden die Startnachrichten der Aktivitäten in der Queue bearbeitet.

Wenn ein Endzustand einer Aktivität erreicht ist, werden die Kontrollflußkonnektoren und Datenflußkonnektoren ausgewertet. Bei der Auswertung des Datenflusses werden die Variablen der Outputcontainer auf Variablen der Inputcontainer einer nachfolgenden Aktivität abgebildet. Anstatt einer 1:1–Kopie kann auch eine Transformation des Wertes über eine Berechnungsvorschrift erfolgen. Über das Verfolgen der ausgewerteten Kontrollkonnektoren werden die nächsten zu startenden Aktivitäten bestimmt.

Erfüllt eine Aktivität die Startbedingung nicht und geht in den Zustand TERMINATED über, tritt die sogenannte *Dead–Path–Elimination* in Aktion. Dabei werden alle Kontrollkonnektoren, die von dieser Aktivität ausgehen, zu falsch evaluiert. Wenn dadurch weitere Aktivitäten in den Zustand TERMINATED überführt werden, wird dieser Vorgang rekursiv fortgesetzt.

Der Start eines Blockes oder eines Subprozesses bedeutet die Instanziierung der untergeordneten Prozeßebene. Da der Block auch als Schleifenkonstrukt benutzt wird, findet bei jedem Schleifendurchgang eine Neuinstanziierung der Prozeßebene statt.

6.2.1. Die Transaktionsgrenzen innerhalb und außerhalb von Sphären

In Abbildung 6.8 sind die vier Fälle angedeutet, wie die Engine Transaktionsgrenzen in Abhängigkeit von den Einträgen in der Warteschlange setzt.

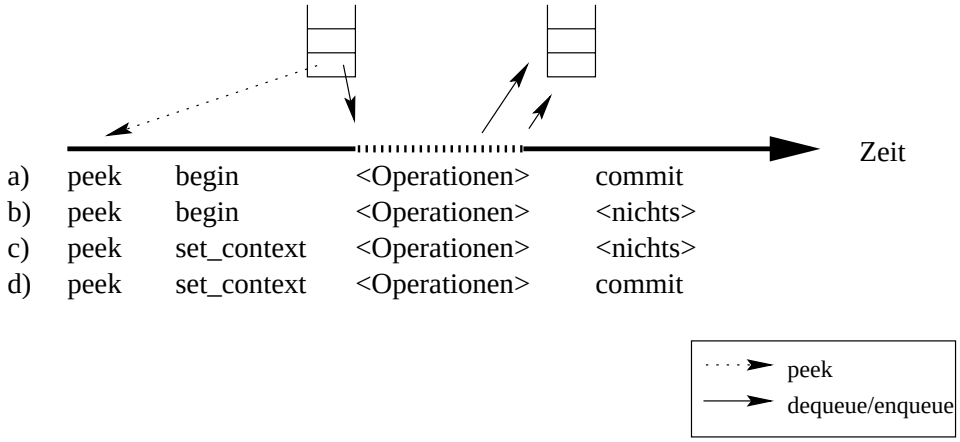

Abbildung 6.8.: Vier Fälle wie auf interne Ereignisse reagiert werden kann

a) Dies ist der Normalfall. Die Engine erfährt durch `peek`, daß der Eintrag in der Warteschlange bestätigt ist (Das heißt, daß die Transaktion, in deren Rahmen der Eintrag in die Warteschlange erfolgt ist, mit „commit“ beendet worden ist). Sie beginnt eine neue Transaktion für die Abarbeitung der Nachricht. In dieser neuen Transaktion wird der Eintrag gelesen und die Engine reagiert entsprechend auf das interne Ereignis. Dabei können weitere interne Ereignisse durch die Engine in die Warteschlange geschrieben werden. Am Ende der Bearbeitung wird die Transaktion bestätigt.

b) Dieser Fall tritt dann auf, wenn das interne Ereignis den Start einer Sphäre zur Folge hat. Die Transaktion wird daher nach dem Start der Sphäre nicht beendet. Die Abarbeitung eines internen Ereignisses geht normal bei `peek` weiter.

c) Der Eintrag in der Warteschlange gehört zu einer noch laufenden Transaktion. Der Eintrag des Ereignisses in die Warteschlange ist also im Rahmen einer Sphärentransaktion erfolgt. Alle weiteren Operationen der Engine in Reaktion auf dieses Ereignis müssen daher in demselben Sphärentransaktionskontext stattfinden. Die Engine bindet sich durch `set_context` an die Sphärentransaktion und führt die Operationen in diesem Kontext aus. Wenn die Sphäre am Ende der Abarbeitung der Nachricht noch nicht beendet worden ist, wird die Transaktion auch nicht bestätigt.

d) In diesem Fall gehört die Nachricht zu einer Sphäre, die in diesem Bearbeitungsschritt beendet wird. Die Sphärentransaktion wird daher am Schluß bestätigt.

6.3. Der Programm–Pool–Manager

Die Implementierung des Programm–Pool–Managers ist in [Ros96] ausführlich beschrieben.

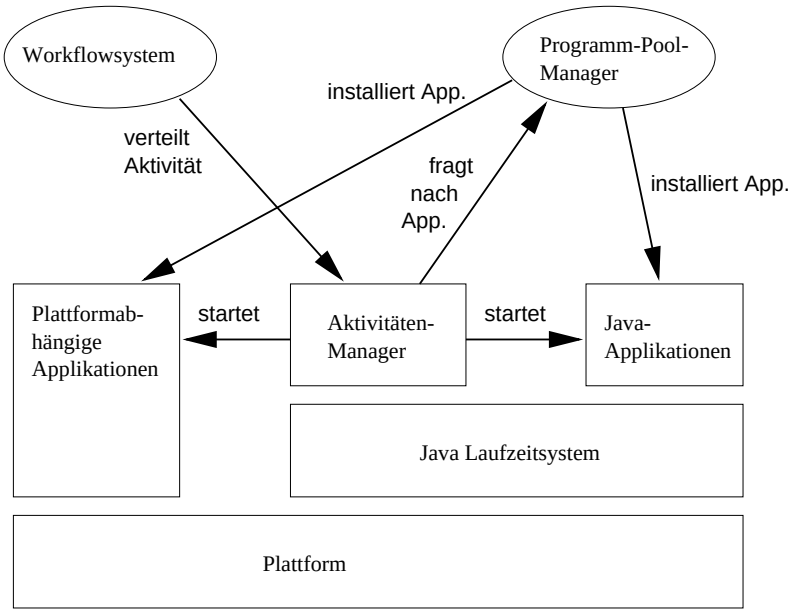

Abbildung 6.9.: Prinzip der Verteilung von plattformunabhängigen Programmen innerhalb eines Workflowsystems

Zur Realisierung von plattformunabhängigen Anwendungsprogrammen in Workflowsystemen werden Java–Applikationen verwendet. Dazu wird auf jedem Arbeitsplatzrechner ein Java–Laufzeitsystem (Java–Interpreter, virtuelle Maschine und Java–API) installiert, das eine plattformunabhängige Schnittstelle zu den Java–Applikationen bietet. Das Laufzeitsystem selbst ist plattformabhängig und muß auf jedes neue System portiert werden (wenn dies noch nicht geschehen ist). Mit diesem Laufzeitsystem als Basis können Java–Applikationen ausgeführt werden (Abbildung 6.9). Plattformabhängige Anwendungsprogramme können ebenfalls ausgeführt werden. Diese setzen dann direkt auf der Plattform auf. In der Abbildung 6.9 wird der Ablauf bei der Bearbeitung einer Aktivität skizziert. Das Workflowsystem weist dem Aktivitäten-Manager eine Aktivität zu. Wenn die für die Bearbeitung benötigte Applikation eine Java–Applikation ist, fordert der Aktivitäten-Manager den Programmcode der Java–Applikation aus dem Programm–Pool an. Der Programm–Pool–Manager überträgt den Programmcode an den Aktivitäten-Manager und dieser führt die Applikation aus. Das System besteht aus folgenden Systemkomponenten:

- Das Java-Laufzeitsystem besteht aus einer virtuellen Maschine (Java–VM), dem Java–Interpreter und dem Java–API, das die plattformunabhängige Schnittstelle zu den Funktionen des Betriebsystems realisiert. Seine Aufgabe ist es, Java–Applikationen auszuführen und die Kommunikation mit der darunterliegenden Plattform zu führen.
- Plattformunabhängige Anwendungsprogramme sind Java–Applikationen, die von der Java–VM ausgeführt werden. Sie verwenden die Funktionen des Be-

triebssystems mit Hilfe einer plattformunabhängigen Schnittstelle und können deshalb, ohne Rekompilation von jedem Java–Laufzeitsystem ausgeführt werden.

- Der Aktivitäten–Manager des Workflowsystems wird ebenfalls als Java–Applikation implementiert, wodurch er auf jedem Arbeitsplatzrechner eingesetzt werden kann, der über ein Java–Laufzeitsystem verfügt. Außerdem wird der Aktivitäten–Manager dazu verwendet, die Java–Applikationen, die für die Bearbeitung von Aktivitäten notwendig sind, vom Programm–Pool–Manager anzufordern und auf einem Massenspeicher des Arbeitsplatzrechners zu installieren. Der Aktivitäten-Manager kann auch plattformabhängige Anwendungsprogramme starten.

- Der Programm–Pool–Manager verwaltet die Applikationen und deren Attribute, wie z. B. Plattform und Version. Der Aktivitäten–Manager kann den Programmcode einer Applikation anfordern, wenn diese für die Bearbeitung einer Aktivität benötigt wird.

- Plattformabhängige Anwendungsprogramme sind Anwendungen, die als ausführbares Programm in der Maschinensprache des Arbeitsplatzrechners vorliegen. Diese Anwendungen verwenden Funktionen des darunterliegenden Betriebssystems bzw. der darunterliegenden Plattform direkt. Sie können ohne das Java–Laufzeitsystem ausgeführt werden, müssen jedoch beim Wechsel des Betriebssystems bzw. des Rechners portiert werden.

6.4. Der Aktivitäten–Manager

Der Aktivitäten–Manager ist die Benutzerschnittstelle des Bearbeiters zum Workflowsystem. Durch das Einloggen des Bearbeiters in das Surro Workflowsystem über den Aktivitäten–Manager nimmt die Person an der Bearbeitung von Workflows teil. Der Aktivitäten–Manager fragt die Arbeitsliste des Bearbeiters nach Aktivitäten ab, die bearbeitet werden sollen, und stellt sie dar. Das Workflowsystem kann jederzeit neue Aktivitäten auf die Arbeitsliste des Bearbeiters legen. Der Aktivitäten–Manager verwaltet drei Listen auf seiner grafischen Oberfläche (siehe Abbildung 6.10). In einer Liste werden alle Aktivitäten dargestellt, die zur Bearbeitung durch den Bearbeiter anstehen, also die eigentliche Arbeitsliste. Der Benutzer kann sich verschiedene Sortierungen der Aktivitäteneinträge und zusätzliche Informationen über die Aktivitäten anzeigen lassen. In einer weiteren Liste werden alle Aktivitäten angezeigt, die momentan von dem Bearbeiter bearbeitet werden. Aktivitäten in der ersten Liste können durch den Benutzer entweder über Doppelklick auf den Eintrag oder über die Menüleiste gestartet werden. Der Aktivitäten–Manager startet dann das zur Aktivität gehörige Anwendungsprogramm. Mit diesem Programm soll der Bearbeiter die Aufgabe erledigen, die der gestarteten Aktivität zugrunde liegt. Nach Beendigung des

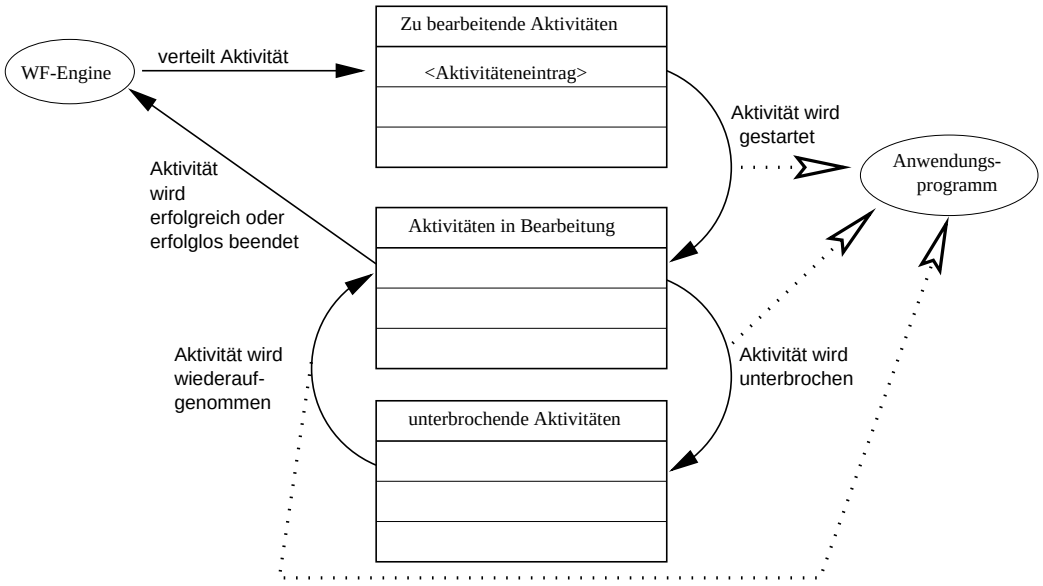

Abbildung 6.10.: Die drei Arbeitslisten des Aktivitäten–Managers

Anwendungsprogramms muß der Bearbeiter entscheiden, ob der Workflow–Engine eine erfolgreiche oder erfolglose Bearbeitung der Aktivität gemeldet werden soll. Falls der Bearbeiter eine erfolgreiche Bearbeitung meldet, prüft die Engine die Nachbedingung der Aktivität. Falls diese zu wahr evaluiert werden kann, dann ist die Aktivität vollständig bearbeitet und der Kontrollfluß des Workflows wird weiterverfolgt. Falls sie zu falsch evaluiert wird, dann bekommt der Bearbeiter eine Meldung, daß die Nachbedingung der Aktivität nicht erfüllt sei. Der Bearbeiter hat dann Gelegenheit zur Nachbesserung.

Die dritte Aktivitätenliste nimmt Einträge von Aktivitäten auf, die während der Bearbeitung durch den Bearbeiter unterbrochen worden sind. Bei entsprechender Unterstützung durch die Anwendung schickt der Aktivitäten–Manager der Anwendung eine Nachricht, sie solle ihren Zustand sichern. Wenn die Bearbeitung im Aktivitäten–Manager wieder aufgenommen wird, wird die Anwendung erneut mit den gespeicherten Zustand gestartet. Diese Funktionalität kann bei längerfristigen Arbeitsunterbrechungen, wie z. B. der Feierabend, sinnvoll angewendet werden. Sie könnte sogar für die Wiedervorlage einer Aktivität benutzt werden. indem nach einem einstellbaren Zeitraum die Bearbeitung der Aktivität wieder aufgenommen wird.

Neben den Funktionen zur Verwaltung der Aktivitätslisten, besitzt der Aktivitäten–Manager noch weitere Funktionalität. Er erlaubt dem Bearbeiter, das Vorgangsinformationssystem aufzurufen, damit er sich grafisch über den Stand der Bearbeitung des gesamten Workflows informieren kann. Über den Aktivitäten–Manager kann der Bearbeiter auch Prozesse starten, und er hat die Möglichkeit, das Auftreten externer Ereignisse[1] der Workflow–Engine zu melden.

[1] siehe Abschnitt 3.1.3

7. Erfahrungen und Ergebnisse

In diesem Kapitel werden die Erfahrungen bei der Implementierung von Surro und die Ergebnisse, die aus dem Implementierung erfolgen, vorgestellt.

7.1. Erfahrungen bezüglich der Entwicklungsumgebungen

Auf der OS/2–Seite sind in erster Linie Erfahrungen mit der Programmierung bzw. Anwendung der CORBA–Implementierung DSOM, der dazugehörigen Implementierung des Object Transaction Service (OTS) und der Datenbank DB2 für OS/2 gemacht worden. Auf der UNIX–Seite wurden Tcl/Tk und Java zusammen mit der Datenbank mSQL eingesetzt.

7.1.1. Implementierung unter DSOM

Die CORBA–Implementierung DSOM *Distributed System Object Model* liegt zur Zeit in der Version 3.0 im Beta–Stadium vor. Die Grundfunktionalität als Object Request Broker, soweit im Prototyp verwendet, bereitet keine besonderen Probleme. Insgesamt aber ist der Beta–Zustand unübersehbar. Dies äußert sich in Surro u. a. dadurch, daß Programmfehler noch häufig ein Reboot erzwingen oder Probleme auftreten und wieder verschwinden, ohne daß dafür Ursachen auszumachen wären.

Die Implementierung mit DSOM und C++ hat sich als insgesamt nicht problemlos herausgestellt. Es fehlt hier noch deutlich eine Verbesserung der Entwicklungsumgebungen. Da CORBA eine komplexe Systemarchitektur darstellt, erweist sich die Einarbeitung und Benutzung als aufwendig und schwierig. Das Paket ist insgesamt äußerst umfangreich bezüglich Platten– und Hauptspeicherbedarf, was sich auch deutlich auf die Performance auswirkt.

7.1.2. Implementierung mit OTS

Der Object Transaction Service als Teil von DSOM wird in Surro sehr intensiv verwendet. Die Verwendung hat sich leider als ein schwieriges Unterfangen herausgestellt. Aufzuführen sind dabei unter anderem folgende Punkte:

- Die Dokumentation ist nicht ausreichend. Es wird im Großen und Ganzen nur ein einziges Beispiel und eine Beschreibung der Schnittstellen gegeben. Dies führt dazu, daß eine große Anzahl an ungeklärten Fragen mittels „trial and error“ evaluiert werden muß, was außerordentlich viel Zeit kostet.

- Insbesonders unzureichend dokumentiert ist die Einbindung von XA Resource–Managern (in diesem Falle die DB2, s.u.) in OTS–gesteuerte Transaktionen.

- Die Performance des Systems ist in der Initialisierungs– und Deinitialisierungsphase (DSOM–Init, Erstellung und Zerstörung der Objekte zur Transaktionsverwaltung) inakzeptabel langsam.

- OTS bzw. DSOM bietet wenig Unterstützung zur Implementierung von transaktionalen Objekten als Resource–Manager an. Nützlich wären insbesondere vorgefertigte Methoden zur Verwaltung von verschiedenen Transaktionskontexten in einem Objekt und zur Persistenz. Ein Ansatz ist die enge Zusammenarbeit mit dem Persistence Service, was zwar laut Dokumentation vorhanden, aber (noch) nicht vollständig bzw. verwendbar implementiert ist.

7.1.3. Das Zusammenspiel von DB2 und OTS

Die verwendete DB2 2.1.1 unter OS/2 ist ein transaktionales Datenbanksystem mit umfangreicher Funktionalität. Massive Einschränkungen in der Benutzbarkeit und grundsätzliche Probleme treten im Rahmen von Surro aber im Zusammenspiel mit OTS auf. Es ist nicht dokumentiert, über welche Schnittstellen der Zugriff auf die DB2 bei Verwendung eines externen Transaktions–Managers wie OTS verwendet bzw. nicht verwendet werden kann. Es ist nicht bzw. nicht ausreichend dokumentiert, welche Besonderheiten dabei zu beachten sind. Dies muß alles zeitraubend evaluiert werden.

Eine starke Einschränkung stellt dabei das Fehlen einer Multithread–Unterstützung dar. Es ist offensichtlich grundsätzlich nicht möglich, aus einem Prozeß heraus von mehreren Threads unter Kontrolle von OTS auf die Datenbank zuzugreifen. Dies führt dazu, daß in Surro *ein einziger Thread alle Datenbankzugriffe unter wechselndem Transaktionskontext durchführen muß.* Der Versuch, von verschiedenen Prozessen anstatt von Threads aus auf die Datenbank zuzugreifen, führte leider ebenfalls zu instabilem Verhalten, wofür die Ursache aber nicht näher geklärt werden konnte.

Der Single–Thread–Zugriff in Verbindung mit dem Wechseln des Transaktionskontextes bei mehreren offenen Transaktionen resultiert in der Problematik, daß jedes Auflaufen auf eine Sperre auf der Datenbank den einen Thread blockiert, der als einziger die Sperre wieder aufheben könnte. Damit dieses Problem nicht zur Wirkung kommt, muß ausgeschlossen werden können, daß es Zugriffe auf sphärenbezogene Daten von Außerhalb einer Sphärentransaktion gibt. Sphärenbezogen heißt dabei, daß diese Daten potentiell von einer Sphärentransaktion angefaßt und somit gesperrt

werden können. Es hat sich in Surro gezeigt, daß das verwendete Datenmodell diese Eigenschaft grundsätzlich bietet, aber man trotzdem häufig auf die Sperrproblematik aufläuft. Dies liegt daran, daß SQL–Anfragen ohne Indexzugriff erst alle Tupel einer Relation lesen müssen, um die qualifizierenden Tupel zu finden. Wenn nun — wie im Datenmodell von Surro der Fall — sphärenbezogene Daten verschiedener Sphären und auch nicht–sphärenbezogene Daten in derselben Relation liegen, läßt sich das Sperrproblem nicht vermeiden.

Die Konsequenz daraus ist die Einführung eines zusätzlichen Parameters für die Zugriffsfunktionen auf die DB2 in der Engine–DB2–Schnittstelle. Der Parameter bewirkt bei Lesezugriffen eine Herabsetzung des Sperrmodus und verhindert jegliches Auflaufen auf eine Sperre. Die Anwendung muß selbst dafür sorgen, daß solcherlei "dirty read" keine Inkonsistenzen bewirkt. In Surro wird dies eingehalten.

Fazit ist, daß die Kombination OTS–DB2 mit gewissen Einschränkungen die Funktionalität liefert, die benötigt wird, daß aber durch die Sperrproblematik des Single–Thread–Zugriffs eigentlich unnötige Komplexität und unnötige Risiken in das System hineingetragen werden.

7.1.4. Implementierung mit Java

Bisher ist nur ein kleiner Teil des Workflowsystems (der Aktivitäten–Manager) in Java programmiert. Durch die zunehmende Bedeutung der Sprache in Internet– und Intranetanwendungen und aufgrund der Eigenschaft der Plattformunabhängigkeit erscheint es sinnvoll, weitere Teile des Systems in Java zu programmieren. Folgende Kritikpunkte, gewonnen aus der Erfahrung mit Surro, stehen einer intensiveren Verwendung im Wege:

- Es fehlen bisher noch ausgereifte Entwicklungsumgebungen. Momentan kommen aber laufend neue Entwicklungsumgebungen auf den Markt, so daß damit gerechnet werden kann, daß in naher Zukunft dieser Kritikpunkt entfällt.

- Der Eigenschaft der Plattformunabhängigkeit gehört zu den großen Pluspunkten der Sprache Java. Allerdings ergeben sich im Detail immer noch Unterschiede auf den verschiedenen Plattformen. Hier ist zu hoffen, daß sich die Implementierungen im Laufe der Zeit stärker angleichen. Ein Nachteil der Plattformunabhängigkeit ist darin zu sehen, daß verschiedene Fähigkeiten der unterschiedlichen Plattformen nicht genutzt werden können.

- Das von Java angebotene Framework für die Erstellung von grafischen Oberflächen (Abstract Window Toolkit, AWT) bietet zuwenig Widget–Klassen mit oftmals nur geringer Funktionalität an. An dieser Stelle mußte Rücksicht auf die unterschiedlichen Fähigkeiten der GUIs auf den verschiedenen Plattformen genommen werden. Daher sind nur Funktionen aus der Schnittmenge der verschiedenen GUIs vorhanden. Die Programmierung mit dem AWT erfolgt noch,

verglichen mit Tk, auf einer relativ niedrigen Ebene. Der Einsatz von Interface Buildern und neue Versionen von AWT können dieses Problem reduzieren.

- Java bietet zum jetzigen Zeitpunkt noch relativ wenig Schnittstellen nach außen, wie z. B. zu einer Datenbank, oder zur Kommunikation zwischen verteilten Objekten an. Durch den Einsatz der CORBA Architektur von der OMG und der Standardisierung eines IDL–to–Java Mappings ist eine Besserung dieses Zustandes zu erwarten.

- Die Performance des implementieren Java Programms ist nicht sehr hoch. Es ist zu erwarten, daß bei komplexeren Programmen dieser Nachteil verstärkt auftritt. Hier bleibt abzuwarten, ob technische Verbesserungen, wie z. B. ein Just–in–Time Compiler, die Laufzeiteigenschaften verbessern kann.

7.1.5. Implementierung mit Tcl/Tk

Die Programmiersprache Tcl/Tk wurde als Implementierungssprache ausgewählt, weil sie sich für die schnelle und einfache Erstellung eines Prototypen eignet. Die Interpretation von Tcl/Tk–Skripten und die Typfreiheit erleichtern Änderungen und halten den Programmcode knapp. Der Entwicklungsprozeß wird so verkürzt. Tcl/Tk bietet mächtige Befehle für die Erstellung von grafischen Oberflächen und für die Interprozeßkommunikation an. Der einfache Zugriff auf relationale Datenbanken hat sehr zum erfolgreichen Einsatz dieser Sprache beigetragen.

Die interpretative Arbeitsweise führt dazu, daß die Skalierbarkeit der Engine, die in dieser Sprache implementiert ist, nicht mehr gegeben ist. Einige wenige Workflows können parallel von der Engine bearbeitet werden, es ist aber wahrscheinlich, daß bei einer größeren Belastung die Arbeitsgeschwindigkeit nicht mehr ausreicht. Tcl/Tk ist eine prozedurale Programmiersprache. Es gibt zwar mehrere objektorientierte Erweiterungen, aber hier konnte sich noch keine Erweiterung durchsetzen. Aus diesen Grund wurde gegen eine objektorientierte Entwicklung der Engine entschieden. Als ein großer Nachteil erweist sich eine fehlende Verbindung zwischen der CORBA Architektur und Tcl/Tk. Es gibt kein standardisiertes Mapping der IDL–Datentypen und Tcl–Datentypen (Strings und assoziative Arrays). Auch gibt es keine IDL–Compiler, die eine Tcl Schnittstelle erzeugen würden. Aus diesem Mangel heraus muß auf die verfügbaren Sockets zurückgegriffen werden, wenn mit Prozessen kommuniziert werden muß, die nicht über die eingebaute Tcl–Interprozeßkommunikation erreicht werden können.

7.2. Workflow–Transaktionen

Das vorgestellte Konzept der Workflow–Transaktionen ist mit vertretbarem Aufwand zu realisieren. Dies hat die Implementierung des Surro Prototypen gezeigt. OTS kann dabei die Rolle des Transaktionsmanagers übernehmen. Alle auftretenden Problemen

konnten gelöst werden, wenngleich daraus einige Einschränkungen entstanden sind. Eine der Einschränkungen ist, daß nur mit einem einzigen Thread und einem Prozeß auf die DB2 Datenbank zugegriffen werden kann, wenn OTS als externer Transaktionsmanager benutzt wird. Als weitere Einschränkung muß hingenommen werden, daß beim Zugriff auf die Workflow–Verwaltungsdaten die von Transaktionen garantierte Isolation durchbrochen werden muß. Aus der Anwendungslogik der Engine heraus kann aber garantiert werden, daß keine inkonsistenten Daten unbestätigter Transaktionen gelesen oder verändert werden.

7.3. Kompensations–Sphären

Das ebenfalls vorgestellte Konzept der Kompensations–Sphären ist mit geringem Aufwand zu realisieren. Weiterführende Versuche mit unterschiedlichen Reihenfolgen der Ausführung der Kompensationsaktivitäten sind aus Zeitgründen nicht durchgeführt worden.

7.4. Kritik am FlowMark Workflow Modell

Das FlowMark Workflow Modell wurde übernommen, um eine weitgehende Übertragbarkeit der Ergebnisse von Surro auf FlowMark zu gewährleisten. Es hat sich aber gezeigt, daß dieses Modell an mehreren Stellen noch Schwächen hat. Daher wurde in Surro versucht, diese Schwächen zu beseitigen, wenn sich dies ohne großen Änderungsaufwand erreichen ließ.

- Im Modell sind nur „lokale“ Variablen für Aktivitäten vorhanden, in den sogenannten Containern. Es kostet aber bei typischen Prozessen viel Modellierungsaufwand die immer wieder benötigten Variablen durch die einzelnen Container „durchzuschleusen“. Eine Art globaler Datencontainer, der „globale Variablen“ enthält, wäre wünschenswert. Die Synchronisation dieser Variablen muß automatisch erfolgen, d. h. es gibt zu einem Zeitpunkt genau einen gültigen Wert.

  Dieser Ansatz wurde nicht implementiert.

- Die Startbedingung für Aktivitäten ist nicht flexibel genug. Alle eingehenden Kontrollflußkonnektoren können entweder alle nur mit AND oder OR verknüpft werden. Im Surro Workflow Modell kann hier eine beliebige logische Verknüpfung der eingehenden Kontrollflußkonnektoren als Startbedingung angegeben werden.

- Das FlowMark zugrundeliegende Organisationsmodell ist sehr einfach aufgebaut. Die Verteilungsstrategie einer Aktivität an einen Bearbeiter ist daher wenig flexibel. In Surro wurde daher ein komplexer aufgebautes Organisationsmodell verwendet.

- FlowMark bietet kein Konzept zur Reaktion auf externe Ereignisse an. In Surro kann jede Aktivität auf das Auftreten eines externen Ereignisses oder eines zusammengesetzten Ereignisses warten, bevor sie ausgeführt wird.
- In Surro existiert im Gegensatz zu FlowMark ein Konzept, mit dem auch plattformunabhängige Programme am Arbeitsplatzrechner automatisch installiert und ausgeführt werden können.

A. Die erstellte Software

A.1. Der Beispielprozeß „Beschwerde über Kreditkartenabrechnung“

In Abb. A.1 ist ein Prozeß dargestellt, mit dessen Hilfe man den Einsatz von Workflow–Transaktionen motivieren kann. Dargestellt ist ein Prozeß aus dem Bankenbereich. Er modelliert den Vorgang, der durch eine Kundenreklamation bei fehlerhafter Abrechnung einer Kreditkarte abläuft. Der Prozeß ist an einen realen Ablauf angelehnt, besitzt natürlich dadurch Unterschiede, daß zur Zeit keine Workflowsysteme mit Workflow–Transaktionen oder vergleichbaren Konstrukten existieren, diese aber hier verwendet werden. Aus Gründen der Übersichtlichkeit sind Datenfluß, Bearbeiter und andere Details nicht abgebildet.

Der Prozeß wird durch einen Anruf des betroffenen Kunden bei der für ihn zuständigen lokalen Hausbank initiiert. Der Sachbearbeiter, der den Anruf entgegennimmt, ermittelt die Art des Vorfalls und startet den entprechenden Prozeß. In der ersten Aktivität *Beschwerde entgegennehmen* werden die Daten des Kunden aufgenommen. Anschließend werden die Belege vom Kunden angefordert, die zur Klärung des Vorfalles notwendig sind *(Belege anfordern)*. Der nächste Schritt besteht darin, den Vorfall anhand der entsprechenden Belege zu überprüfen. Der Prozeß kann somit so lange nicht fortgeführt werden, bis die Belege vorliegen. Dies wird dadurch sichergestellt, daß die Aktivität *Belege prüfen* auf das Eintreten des externen Ereignisses „Kunde schickt Belege“ wartet und vorher nicht aktiv wird. Das Ereignis muß von außen in das System eingebracht werden.

Sind die Belege vorhanden, wird die Aktivität gestartet. Kommt der Prüfer zum Ergebnis, daß die Reklamation unbegründet ist, wird die Beschwerde sofort abgelehnt (unterer Pfad des Graphen), der Kunde wird benachrichtigt *(Beschwerde ablehnen)*, der Vorfall wird archiviert und der Prozeß beendet. Wird die Reklamation akzeptiert, muß zum einen diese Tatsache zusammen mit allen relevanten Informationen im lokalen Informationssystem gesichert werden *(Eintrag in lokale DB)*, zum anderen muß dem betroffenen Kreditkarteninstitut dieselbe Information verfügbar gemacht werden *(Eintrag in DB Karteninstitut)*. Letzteres stößt beim Kreditkarteninstitut wiederum einen Prozeß an, der im Endeffekt dazu führen soll, daß der Fehlbetrag an die lokale Bank überwiesen wird (In Abb. A.1 nicht näher dargestellt). Dieses „Verfügbarmachen“ der Informationen erfolgt sinnvollerweise dadurch, daß über einen entfernten

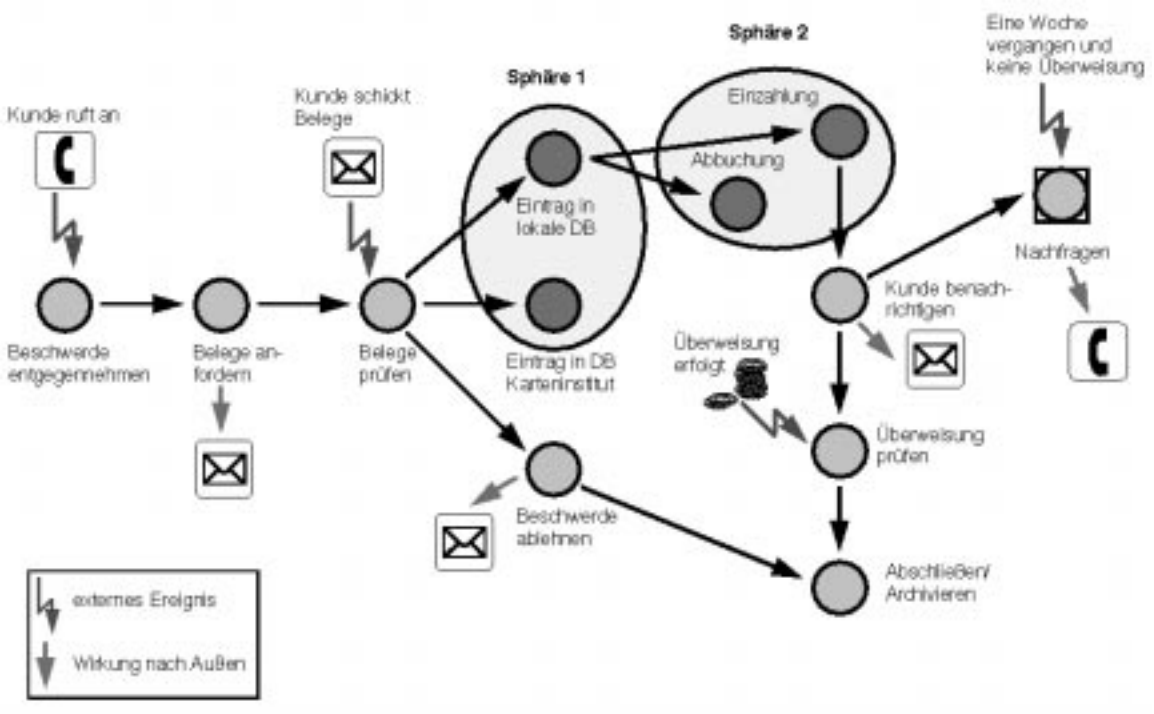

Abbildung A.1.: Beispielprozeß „Kundenbeschwerde bei fehlerhafter Kreditkartenabrechnung“

Datenbankzugriff die Daten direkt in die Datenbank des Karteninstituts eingetragen werden. Damit nun keine Inkonsistenzen zwischen dem lokalen und entfernten Datenbestand entstehen können, ist es notwendig, die beiden Aktivitäten *Eintrag in lokale DB* und *Eintrag in DB Karteninstitut* in einer verteilten Transaktion zusammenzufassen. Dazu dient die erste Sphäre.

Ist dies erfolgt, wird dem betroffenen Kunden der genehmigte Fehlbetrag von einem speziellen Ausgleichskonto der lokalen Bank sofort überwiesen, da die Bearbeitung durch das Karteninstitut normalerweise länger als für den Kunden akzeptabel dauert. Dazu muß vom Ausgleichskonto der Betrag abgebucht und dem Kundenkonto gutgeschrieben werden. Diese beiden Aktivitäten sind wiederum in einer Sphäre zusammengefaßt und somit durch eine Workflow–Transaktion geschützt. Dies erlaubt auch die Parallelisierung der Aktivitäten *Einzahlung* und *Abbuchung*, da diese jetzt als atomare Einheit gesehen werden können.

Hier ist die Notwendigkeit eines transaktionalen Schutzes natürlich offensichtlich, da ohne Schutz durch Fehler beliebige Inkonsistenzen auftreten können, wodurch eventuell Geld vernichtet bzw. erzeugt würde, oder auch Buchungen doppelt erfolgen könnten.

Nach Beendigung der Sphäre wird der Kunde über die Buchung des Fehlbetrages auf sein Konto informiert *(Kunde benachrichtigen)*. In *Überweisung prüfen* wird die Überweisung des Fehlbetrages vom Karteninstitut an die lokale Bank auf Korrektheit

überprüft. Für den Fall, daß die Prüfung einen Fehler ergibt, ist im Prozeß aus Einfachheitsgründen keine weitere Vorgehensweise definiert.

Die Aktivität *Überweisung prüfen* kann aber erst dann startbar werden, wenn die Überweisung durch das Karteninstitut tatsächlich erfolgt ist. Somit wird, wie bei *Belege prüfen*, auf ein externes Ereignis *(Überweisung erfolgt)* gewartet. Da dies unter Umständen sehr lange dauern kann, wird ein Schleifenkonstrukt (Block *Nachfragen*) eingeführt. In diesem Block befindet sich eine einzige Aktivität (nicht in Abb. A.1 dargestellt), die jeweils nach einem bestimmten Zeitraum (hier: eine Woche) aktiviert wird, falls die Überweisung noch nicht erfolgt ist. In dieser Aktivität kann dann beim Karteninstitut nachgefragt werden, wie die Sachlage ist, etc.

In der letzten Aktivität wird dann der gesamte Vorgang abgeschlossen und archiviert.

A.2. Einschränkungen der aktuellen Implementierung (Stand Ende 1996)

- Es gibt keinen graphischen Editor zur Spezifikation von Workflows.
- Es gibt keine Komponente mit der laufende Workflow–Instanzen in der Art von Ad–hoc–Modifikationen verändert werden können.
- Der Knotentyp Unterprozeß ist nicht implementiert.
- Der Knotentyp Block darf nicht in Sphären verwendet werden.
- Das Erzeugen von Timeouts und die Reaktion auf Timeouts sind nicht realisiert.
- Zusammengesetzte Ereignisse sind nicht im vollen Umfang implementiert.
- Aktivitäten können nicht mehreren Personen gleichzeitig auf ihrer Arbeitsliste angeboten werden.
- Ersatzaktivitäten sind nicht vollständig implementiert.

A.3. Die Softwaremodule

In den Tabellen A.1 bis A.3 werden die im Workflowsystem Surro implementierten Softwaremodule aufgelistet. In Tabelle A.1 sind die Module aufgeführt, die in Tcl/Tk implementiert sind und unter UNIX ausgeführt werden, in Tabelle A.2 die in Java implementierten und somit weitgehend plattformunabhängigen Module, und schließlich in Tabelle A.3 die OS/2–Seite mit ihrem in C++ implementierten Code.

Die in der Tabelle A.4 aufgelistete Software wurde bei der Implementierung des Workflowsystems Surro benutzt. In Tabelle A.5 sind die Bezugsquellen dazu aufgeführt.

Programmmodul	Größe	Autor	Bemerkung
suEngineGUI.tcl	24 KB	Schreyjak	Grafische Oberfläche der Engine
suEngineModul.tcl	117 KB	Schreyjak	Die eigentliche Engine
suWFSManager.tcl	55 KB	Schreyjak	Session Verwaltung und Schnittstelle der Engine zum Aktivitäten–Manager
suQueue.tcl	3 KB	Schreyjak	Queue Verwaltung für mSQL
suSOCKlib.tcl	5 KB	Schreyjak	Routinen für Socket Kommunikation
suOrgModul.tcl	10 KB	Schreyjak	Routinen für Organisationsmodul
suAktManager.tcl	53 KB	Schreyjak	Aktivitäten Manager in der Tcl Version
msqltclproxy.tcl	3 KB	Schreyjak	Datenzugriff auf msql
msqltclproxysocket.tcl	8 KB	Schreyjak	Datenzugriff auf DB2 über sockets
suTCLDBaddon.tcl	5 KB	Schreyjak	Erweiterte Funktionen für Datenzugriff
suADDitional.tcl	4 KB	Schreyjak	Mehrfach verwendete Hilfsfunktionen
suMonitor.tcl	70 KB	Bildstein	Workflow Monitor

Tabelle A.1.: Surro–Module, implementiert in Tcl/Tk

Programmmodul	Größe	Autor	Bemerkung
AcitivityManager.java	70 KB	Rosenauer	Aktivitäten Manager in Java Version
ProgPool.java	29 KB	Rosenauer	Programm Pool Verwaltung in Java Version
ComAdapter.java	34 KB	Rosenauer	Kommunikation zum Workflow Session Manager

Tabelle A.2.: Surro–Module, implementiert in Java

Programmmodul	Größe	Autor	Bemerkung
suTAadapter.cpp	21 KB	Bildstein	Kommunikationsschnittstelle zu Engine und Monitor, ruft Funktionen aus db2tcl.cpp auf
suDb2tcl.cpp	60 KB	Bildstein	Zugriff auf DB2, TA–Kontext–Management und transaktionale Queue
suContext.cpp	10 KB	Bildstein	DSOM–Objekt zur Verwaltung der Transaktionskontexte
suTqueue.cpp	23 KB	Bildstein	DSOM–Objekt zur Verwaltung einer transaktionalen Message–Queue
suTqres.cpp	4 KB	Bildstein	Resource–Objekt zur transaktionalen Message–Queue
suPec.cpp	75 KB	Bildstein	OS/2 Program Execution Client zur automatischen Ausführung von transaktionalen Aktivitäten
suAccount.cpp	12 KB	Bildstein	Transaktionales Konto–Objekt
suAccres.cpp	5 KB	Bildstein	Resource–Objekt zum Konto–Objekt

Tabelle A.3.: Surro–Module, implementiert in C/C++ mit DSOM

Software	Was ist es?
mSQL	Datenbank
DB2/2 V2.1.1	kommerzielle Datenbank (IBM)
SOMobjects 3.0	CORBA ORB (IBM)
OTS	CORBA Object Service: Object Transaction Service
CSet++	C/C++ Compiler unter OS/2 (IBM)
TCL 7.5	Programmiersprache
TK 4.1	Programmiersprache
MSQLTCL	mSQL–Anbindung für Tcl/Tk
Java	Programmiersprache

Tabelle A.4.: Verwendete Software

Software	Quelle
mSQL	ftp://ftp.bond.edu.au/pub/Minerva/msql/msql-1.0.16.tar.gz
DB2	IBM-Produkt
SOMobjects	http://www.software.ibm.com/objects/somobjects/
OTS	Bestandteil von SOMobjects 3.0
CSet++	IBM-Produkt
TCL 7.5	http://www.sunlabs.com/research/tcl/
TK 4.1	http://www.sunlabs.com/research/tcl/
MSQLTCL	ftp://ftp.bond.edu.au/pub/Minerva/msql/Contrib/msqltcl-1.50.tar.gz
Java	http://java.sun.com/

Tabelle A.5.: Quellen der verwendeten Software

Literaturverzeichnis

[CM94] Chakravarthy, S. ; Mishra, D.: Snoop: An expressive event specification language for active databases. **In:** *Data & Knowledge Engineering* 14 (1994), November, Nr. 1, S. 1–26

[GR93] Gray, Jim ; Reuter, Andreas: *Transaction Processing*. Morgan Kaufmann, 1993

[Hug96] Hughes, David J.: *mSQL — A Lightweight Database Engine*. Hughes Technologies Pty. Ltd., 1996. – http://Hughes.com.au/

[Ley95] Leymann, F.: Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. **In:** Lausen, G. (Hrsg.): *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft*. Berlin : Springer, Maerz 1995, S. 51–70

[LR94] Leymann, Frank ; Roller, Dieter: Business Process Management With FlowMark. **In:** *Proc. 39th IEEE Computer Society Int. Conference (CompCon)*. San Francisco, Cal., Februar 1994, S. 230–234

[OTS94] Object Management Group (OMG): *Object Transaction Service*. August 1994. – Document No. 94.8.4

[Ous94] Ousterhout, John K.: *Tcl and Tk Toolkit*. Messachusetts : Addision Wesley, 1994

[Ros96] Rosenauer, Hansgeorg: *Entwurf und Implementierung eines Verteilungsmechanismus für plattformunabhängige Anwendungsprogramme in Workflow–Systemen*, Universität Stuttgart, Studienarbeit, 1996.

[SB96] Schreyjak, Stefan ; Bildstein, Hubert: *Fehlerbehandlung in Workflow–Management–Systemen* Universität Stuttgart, Software–Labor, Fakultätsbericht Nr. 1996/17, Software–Labor Bericht SL–3/96.

[Sch93] Schmidt, Ursula: Transaktionskonzepte in der Fertigung. **In:** *Proc. Datenbanksysteme in Büro, Technik, Wissenschaft* Braunschweig, März 1993.

[Sch95] Schreyjak, Stefan: *Anforderungsanalyse von Workflowsystemen*, Universität Stuttgart, Fakultät Informatik, Diplomarbeit, 1995.

[Täu96] TÄUBER, Wolfgang: *Transaktionale Datei- und Dokumentenverwaltung in Workflow–Management–Systemen*, Universität Stuttgart, Diplomarbeit, August 1996. – Diplomarbeit Nr. 1380

[WfM96] WFMC: *Workflow Management Coalition Specification — Terminology and Glossary* http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html 1996.

[X/O91] X/Open Company Ltd.: *X/Open: Distributed Transaction Processing: The XA Specification*. 1991