23. Ein Workflow-Managementsystem auf der Basis aktiver Datenbanken

Johann Eder, Herbert Groiss

Abstract

Dieses Kapitel stellt eine Architektur für die Realisierung von Workflow-Managementsystemen vor, die auf der Verwendung von aktiven Datenbanksystemen basiert. Bei diesem Ansatz werden alle Informationen über das Workflowsystem, das sind Prozeßstrukturen, Benutzer bzw. Rolleninformation, Metadaten sowie Beziehungen zwischen Prozessen und Daten, als auch sämtliche Informationen über die dynamische Prozeßdurchführung in der Datenbank verwaltet. Die Workflowdefinitionen werden auf Trigger übersetzt, die die Prozeßdurchführung ereignisorientiert steuern. Als Beispiel wird der Protoyp Panta Rhei vorgestellt, dem diese Architektur erfolgreich zugrundegelegt wurde.

23.1 Einleitung

Workflow-Systeme unterstützen die Durchführung von Geschäftsprozessen mit informationstechnischen Mitteln. Geschäftsprozesse bzw. Vorgänge werden mit den dafür entwickelten Modellierungswerkzeugen konzeptuell modelliert. Diese konzeptuellen Prozeßmodelle können dann auf Workflowdefinitionen, die Prozeßschemata, abgebildet werden. Es gibt dafür natürlich verschiedenste Ansätze ([HEA90, LCS82, ML91, TLF88, Zlo82, EB82]) mit recht unterschiedlicher Mächtigkeit oder Flexibilität. Im wesentlichen spezifizieren Workflowdefinitionen folgendes:

- Welche Aufgaben (Tasks) ausgeführt werden sollen.
- In welcher Reihenfolge die Aufgaben ausgeführt werden (meist abhängig von Entscheidungen, die ebenfalls Teil des Prozesses sind).
- Wer die Aufgaben ausführen soll - insbesondere, ob die Aufgabe maschinell (durch ein Anwendungsprogramm) oder durch einen Mensch-Maschinen-Dialog ausgeführt werden soll.
- Welche Randbedingungen in bezug auf Zeit, Qualität und Ressourcenverbrauch eingehalten werden müssen.

Workflow-Managementsysteme haben die Aufgabe, Workflowdefinitionen entgegenzunehmen und in ausführbare Prozesse zu transformieren. Sie steuern die Durchführung dieser Prozesse und gewährleisten die Sicherheit und Zuverlässigkeit der Prozeßdurchführung auch bei Hard- und Softwarefehlern. Außerdem verwalten sie Benutzerdaten und Berechtigungen, dokumentieren die Prozeßdurchführung und generieren Daten für das Management der Geschäftsprozesse.

Wir können die Aufgaben eines Workflow-Managementsystems deshalb auf drei Ebenen betrachten. Die erste Ebene verwaltet die Metadaten, das sind die Definitionen von Workflows und ihrer zugehörigen Prozesse, Aktivitäten, Tasks, die von diesen verwendeten Daten, sowie die Beziehung zwischen Prozessen und Daten. Die zweite Ebene umfaßt die Verwaltung der Instanzen, das sind Prozeßinstanzen, Dateninstanzen, Benutzer und Rolleninformationen sowie die Beziehungen zwischen Benutzern, Dateninstanzen und Prozeßinstanzen. Die dritte Ebene beinhaltet die Prozeßsteuerung, d.h das Anstoßen von Prozessen (Aktivitäten, Tasks), die Weiterleitung von Dateninstanzen, der automatische Start von Programmen, etc.

In diesem Kapitel wird ein neuer Ansatz für die Realisierung von Workflow-Managementsystemen beschrieben. Die Grundidee ist die Verwendung einer aktiven Datenbank für die Verwaltung sämtlicher Daten sowie für die Steuerung der Prozeßinstanzen. Aktive Datenbanken erweitern herkömmliche (passive) Datenbanken durch eine dynamische Komponente in Form von Produktionsregeln bzw. Trigger, mit der sie auf Ereignisse wie Änderungen des Datenbankinhaltes automatisch reagieren können. Sie sind deshalb vorzüglich für Applikationen geeignet, die - wie Workflow-Systeme - inhärent daten- und ereignisgesteuert sind.

Für die Verwaltung der Daten und Metadaten werden in Workflow-Managementsystemen häufig Datenbanken herangezogen. In unserem Ansatz wird auch die dynamische Statusinformation der Prozeßinstanzen in den Datenraum abgebildet. Für jede Prozeßinstanz wird in der Datenbank eine Dateninstanz angelegt, die unter anderem den aktuellen Stand der Prozeßinstanz enthält. Aus dieser Prozeßbeschreibung kann insbesondere abgelesen werden, welche Tasks bereits absolviert sind und welche Tasks für die Durchführung bereit stehen. Diese Statusinformation ist persistent, da sie von einer Datenbank verwaltet wird. Änderungen an dieser Statusinformation ist der Transaktionsverwaltung des Datenbanksystems unterworfen.

Für die dynamische Prozeßdurchführung wird ebenfalls die Datenbank, und zwar die Trigger der aktiven Datenbank, herangezogen. Damit wird eine hochintegrierte Architektur realisiert, die auf einem einzigen Basissystem beruht. Gleichzeitig bietet diese Architektur eine größtmögliche Offenheit, da die aktive Datenbank beliebige externe Prozesse starten kann und über die Verwendung der Datenbank als Kommunikationsvehikel Daten zwischen Programmen ausgetauscht werden können. Damit können in einen Workflow beliebige Anwenderprogramme, insbesondere bereits bestehende, eingebunden werden. Außerdem kann die Vielzahl der Schnittstellen und Werkzeuge moderner Datenbanksysteme direkt für das Workflowsystem verwendet werden. Eine Besonderheit bietet dieses Konzept auch für die Auslösung von Workflowprozessen. So kann das Workflowsystem in bestehende, auf Datenbanken basierende Informationssysteme integriert werden, so daß auf bestimmte Situationen, die durch bestehende Applikationen in die Datenbank abgebildet werden, über Trigger automatisch ein Workflow gestartet wird. Dafür braucht die bestehende Applikation in keiner Weise geändert werden.

Die hier vorgestellte Architektur ist auch dadurch charakterisiert, daß möglichst viel der vom aktiven Datenbanksystem zur Verfügung gestellten Funktionalität direkt genutzt wird. So wird zum Beispiel für die Steuerung der Nebenläufigkeit direkt das Transaktionssystem der Datenbank verwendet. Weiters wird die erforderliche Sicherheit durch den Zugriffsschutz der Datenbank direkt gewährleistet. Das Recovery-System der Datenbank sorgt für die Wiederherstellung eines konsistenten Zustandes der Datenbank und damit auch des Workflowsystems im Falle eines Systemausfalls mit Primär- oder Sekundärspeicherverlust. Dabei werden nicht nur die Daten wiederhergestellt, sondern auch alle Prozesse werden auf den letzten bestätigten Stand gesetzt - ohne jeglichen zusätzlichen Implementierungsaufwand. Diese Architektur zeichnet sich daher auch durch vergleichsweise niedrigen Aufwand für die Realisierung eines Workflow-Managementsystems aus.

Im nächsten Abschnitt geben wir eine kurze Einführung in die Funktionsweise von aktiven Datenbanken, danach wird die Abbildung von Workflow-Beschreibungen in Regeln aktiver Datenbanken beschrieben, Abschnitt 23.4 zeigt die Architektur unseres Prototypen Panta Rhei (Heraklit: „Alles fließt“) und Abschnitt 23.5 bringt eine Zusammenfassung und Bewertung unseres Ansatzes.

23.2 Aktive Datenbanken

Konventionelle Datenbanken sind passiv, das heißt, Anfragen und Veränderungen werden nur durch explizite Aktionen eines Benutzers oder Anwendungsprogramms durchgeführt. Aktive Datenbanken (für eine Einführung siehe [Day88], [Cha92]) hingegen erlauben die Spezifikation von Aktionen, die automatisch ausgeführt werden, wenn bestimmte Ereignisse eintreten. Die grundsätzlichen Elemente der Architektur von aktiven Datenbanken sind in Abbildung 23.1 dargestellt.

Das Datenbank-Managementsystem einer aktiven Datenbank enthält einen Mechanismus zum Erkennen von Ereignissen, zum Prüfen von Bedingungen und zum Durchführen bzw. Anstoßen von Aktionen. Bei allen Abfragen und Veränderungen, die in der Datenbank eintreffen, wird überprüft, ob eine Aktion durchgeführt werden soll; diese Aktion kann wieder Veränderungen in

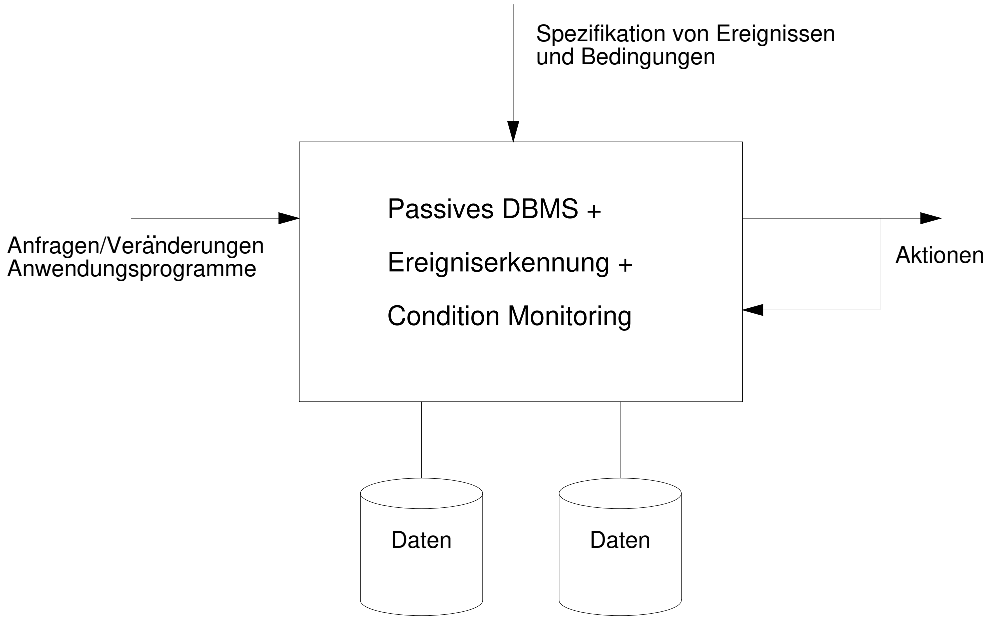

Abbildung 23.1: **Prinzip einer aktiven Datenbank**

der Datenbank auslösen oder eine Aktion außerhalb der Datenbasis anstoßen.

Die Spezifikation von Ereignis, Bedingungen und Aktionen erfolgt deklarativ in Form von Event-Condition-Action (ECA) Regeln.

Jeder Zugriff auf die Datenbank durch einen Benutzer oder ein Anwendungsprogramm wird als Ereignis aufgefaßt, das eine Regelanwendung anstoßen kann. Bei Triggerung einer Regel durch ein Ereignis werden die Bedingungen dieser Regel überprüft. Sind sie erfüllt, werden die Aktionen der Regel ausgeführt.

Bedingungen sind dabei Beschreibungen von Zuständen in der Datenbank. Aktionen können wiederum Veränderungen in der Datenbank sein, aber auch Aufrufe von externen Prozeduren.

Im folgenden wird für Beispiele die Syntax von SQL3 [IA93] verwendet, der - etwas vereinfachte - Aufbau einer Regel sieht folgendermaßen aus:

`create trigger` *name*
`after` *event* `on` *table*
`when` *condition*
action

Mit `create trigger` wird eine Regel *name* definiert, die auf Veränderungen der Tabelle *table* reagiert. Das die Regel triggernde Ereignis wird nach dem Schlüsselwort `after` spezifiziert, der Bedingungsteil nach dem Schlüsselwort `when` enthält eine quantifizierte SQL-Abfrage, die einen Booleschen Wert retouniert. Als Aktionen sind in SQL formulierte Datenbankaktionen zulässig.

Beispiel 23.1 Bei Veränderung des Lagerbestandes unter einen festgesetzten Wert soll automatisch eine Bestellung generiert werden. Dazu wird eine Regel formuliert, die bei Veränderungen der Bestände getriggert wird und beim Unterschreiten des Schwellwertes den Teil in eine Bestelliste einträgt.

```
create trigger store_control
after update of quantity on store
when new.quantity < 5
insert into order_list values (new.art_no, ...);
```

Diese Regel reagiert auf Veränderungen in der Tabelle `store`, das triggernde Ereignis ist eine Veränderung des Attributs `quantity` bei einem Tupel dieser Tabelle. Die Bedingung ist eine SQL-Abfrage, die überprüft, ob die neue `quantity` kleiner `5` ist. Trifft dies zu, wird ein Eintrag in die Tabelle `order_list` durchgeführt.

Den Vorteilen von aktiven Datenbanken stehen natürlich auch gewisse Nachteile gegenüber. Zum einen ist die Verarbeitung der Regeln eine zusätzliche Belastung für den Datenbankserver. Aktive Datenbanken stellen daher größere Anforderungen an die Rechnerleistung der Server. Zum anderen ist die Strukturierung sehr großer Regelmengen noch nicht wirklich zufriedenstellend gelöst.

23.3 Abbildung von Workflow-Modellen auf aktive Datenbanken

23.3.1 Referenzmodell einer Prozeßbeschreibung

Bisweilen gibt es kein allgemein akzeptiertes Metamodell zur Beschreibung von Workflows (obwohl Versuche in diese Richtung gehen, siehe [WMC94b], [EL95]). Abb. 23.2 stellt das Schema eines einfachen Modells dar:

Workflows sind zusammengesetze Aktivitäten, die die Ausführung mehrerer *Tasks* durch verschiedene *Akteure* beschreiben. Ein Akteur ist entweder durch einen konkreten *Benutzer* oder ein Programm, das die Datenmanipulation ausführt, oder eine *Rolle* stellvertretend für eine Menge von Akteuren, spezifiziert. *Forms* sind die Datencontainer, die die Applikations- und Prozeßdaten zwischen den Aktivitäten weiterreichen. Die Wege der Forms werden mittels *Flows* spezifiziert.

Zur Darstellung von Workflows verwenden wir im folgenden die von uns entwickelte Workflow-Beschreibungssprache WDL.

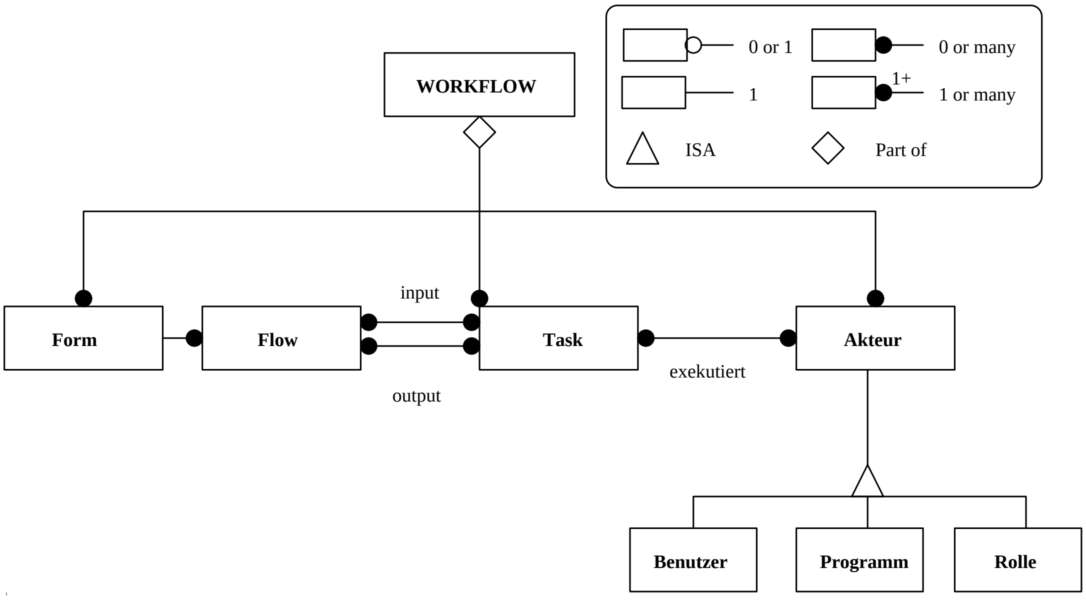

Abbildung 23.2: **Workflow Metamodell**

23.3.2 Die Workflow-Beschreibungssprache WDL

WDL ist im wesentlichen ein Darstellungskonzept für die oben beschriebene Modellierung (Abb. 23.2) von Prozessen. Die wichtigsten Designkriterien für diese Sprache waren: einen einfach zu verwendenden Formalismus zu schaffen, der es ermöglicht eine breite Palette von Geschäftsprozessen zu modellieren, sowie gute Abbilddbarkeit auf eine Implementierung zu gewährleisten.

Die wesentlichen Elemente eines Workflows lassen sich graphisch mittels des WDL Prozeßdiagramms darstellen. Abb. 23.3 zeigt ein solches für den Antrag einer Dienstreise.

Dieser Prozeß erfordert die Interaktion verschiedener Personen und Abteilungen. Ein Antragssteller, der eine Reise plant, braucht die Genehmigung von Institutsvorstand und Dekan. Nach der Reise bekommt er das Geld von der Personalabteilung. Die Tasks werden durch rechteckige Kästchen repräsentiert, in denen der Name des Tasks sowie - in eckigen Klammern - der Ausführende eingetragen wird. Für letzteren wird entweder direkt der Benutzer oder eine Rolle spezifiziert (z.B. Dekan). Für Tasks, die automatisch, d.h. ohne Benutzer-Interaktion ablaufen, wird der Pseudo-Benutzer SYSTEM spezifiziert. Dies erlaubt die Integration von beliebigen Programmen, die die Daten manipulieren. In obigem Beispiel kann ein Vorschlag automatisch generiert werden, der z.B. die Summe des noch vorhandenen

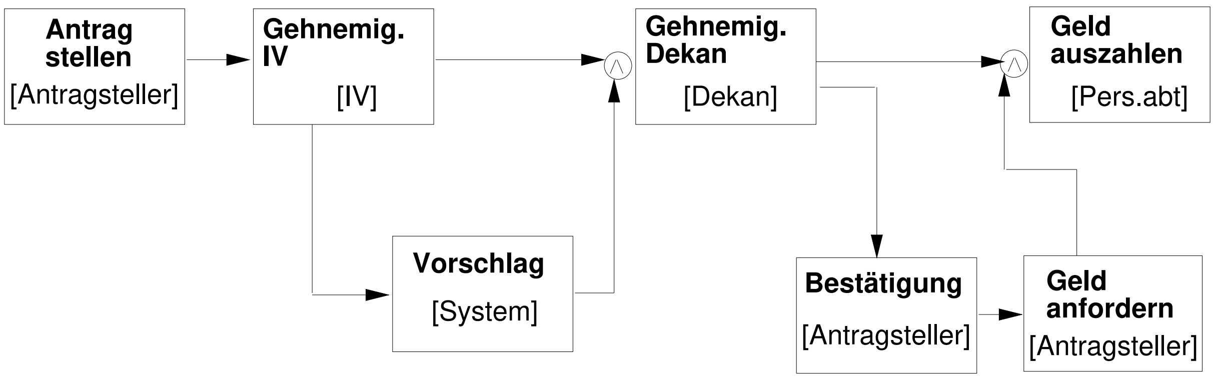

Abbildung 23.3: **Dienstreise**

Geldes abfragt und entsprechende Aktionen setzt. Außerdem kann der Benutzer als DYNAMIC spezifiziert werden; in diesem Fall liest das System den Benutzer aus einem Feld in einem der bearbeiteten Forms aus.

Der Kontrollfluß wird durch die Pfeile repräsentiert. Ein Task kann einen oder mehrere Eingangsflows sowie einen oder mehrere Ausgangsflows haben, die verschieden verknüpft werden können:

Eingangsseite des Tasks:

- mehrere Flows münden in einen Task: dies bedeutet, daß der Task aktiviert wird, wenn eine der Forms den Task erreicht.
- mehrere Inputflows sind logisch verknüpft: Der Task wird aktiviert, wenn die Boolesche Verknüpfung zwischen den Eingangsflows erfüllt ist. z.B.: f1 UND (f2 ODER f3) bedeutet, daß der Task aktiviert wird, wenn die Form f1 und entweder f2 oder f3 eingetroffen sind.
- Synchronisationpunkt: Wenn ein Form zur parallelen Bearbeitung an verschiedene Tasks weitergeleitet wird, muß am Ende der Parallelführung ein Synchronisationspunkt liegen. Der nachfolgende Task wird nur ausgeführt, wenn alle parallelen Teilpfade abgearbeitet worden sind.

Ausgangsseite des Tasks:

- keine Verknüpfung: Alle Forms werden den entsprechenden Nachfolgetasks geschickt.
- UND Verknüpfung: Die Form wird an zwei Tasks zur parallelen Bearbeitung weitergereicht.
- ODER: Je nachdem, welche Flow-Bedingung erfüllt ist, wird die Form zu einem der beiden Nachfolgetasks weitergeschickt.

Zusätzlich zur Definition des Prozeßdiagramms ist es notwendig, einige Informationen zum Prozeß anzugeben:

- generelle Informationen zum Workflow: Definition aller Benutzer und Rollen, die im Workflow vorkommen.
- Struktur der involvierten Forms: Das Prozeßdiagramm zeigt nur den Fluß von Forms an, nicht deren innere Struktur.

zusätzliche Informationen zum Task:

- Postconditions: Durch die Definition von Postconditions erreicht man, daß am Ende eines Tasks definierte Zustände herrschen.
- Prozeduren: Eine before-procedure und eine after-procedure können zur Ausführung bevor bzw. nach der Ausführung eines Tasks spezifiziert werden.
- Benutzerauswahl: Wenn eine Rolle als Akteur bei einem Task spezifiziert wird, erfordert dies die Definition eines Selektionskriteriums, um den konkreten Benutzer zum Ausführungszeitpunkt bestimmen zu können. Mögliche Auswahlkriterien sind: Wähle Benutzer mit der kleinsten workload, zufällige Auswahl, o.ä.
- Zugriffsarten: Der Zugriff der Tasks auf die einzelnen Forms muß spezifiziert werden, so ist es möglich, daß auf einige Felder der Form überhaupt kein Zugriff besteht, auf andere Lesezugriff oder Lese- und Schreibmöglichkeit.

Für die Prozeduren, Pre- und Postconditions wurde keine exakte Syntax definiert, da diese von der Implementierung abhängt. In unserer Implementierung auf der Basis einer SQL-Datenbank sind diese Prozeduren und Ausdrücke jeweils SQL Konstrukte, somit ist eine weitestgehende Portierbarkeit gewährleistet. In Abschnitt 23.4.2 wird ein Designeditor für WDL beschrieben.

23.3.3 Ausführungsmodell

Ein WDL-Prozeßdiagramm definiert Abläufe als den Weg von Forms zwischen Tasks. Die Datenmanipulationen innerhalb der Tasks werden nicht betrachtet. Das Ausführungsmodell für WDL kann daher mit den einzelnen Schritten, die zwischen der Beendigung eines Tasks und dem Starten eines anderen vor sich gehen, beschrieben werden.

Nach dem Abschluß eines Tasks - üblicherweise eine Manipulation von Dokumenten oder Formularen - geht die Kontrolle an den Workflow-Server, der die folgenden Schritte ausführt:

1. Die - optionale - Post-Prozedur wird ausgeführt.
2. Die Bedingungen, die nach der Task-Ausführung gelten müssen, werden überprüft. Wenn sie erfüllt sind, gilt diese Aktivität als abgeschlossen, im anderen Fall erhält der Task eine Nachricht, daß eine Weiterbearbeitung notwendig ist.
3. Entsprechend der Definition der Abläufe werden die bearbeiteten Dokumente an die Nachfolger-Aktivitäten geschickt.
4. Die Vorbedingungen der Tasks werden überprüft.
5. Falls die Vorbedingungen zur Ausführung einer Aktivität erfüllt sind, wird sie einem bestimmten Benutzer (oder einer Maschine) zugeordnet.
6. Eine den Task vorbereitende Prozedur wird ausgeführt.
7. Der Benutzer (das Client-Programm) erhält ein Signal, daß eine Aktivität zur Bearbeitung ansteht.

Für die Abbildung dieses Ablaufs in die Datenbank ist es notwendig, die Statusinformationen zu den laufenden Prozessen in Tabellen abzulegen. Beim Initiieren eines Prozesses wird für die neue Prozeßinstanz eine Identifikation und ein Eintrag in eine entsprechende Tabelle generiert. Der Großteil der Informationen zu einem laufenden Prozeß wird mit den Task-Instanzen gespeichert: eine Identifikation, der zugehörige Prozeß und Task, die Prozeßinstanz, der Benutzer, ein Zeitstempel, sowie der Status der Bearbeitung entsprechend des obigen Ablaufmodells. Außerdem werden alle Form-Instanzen mit zugehöriger Prozeßinstanz, dem Typ der Form und eine Referenz zu den Daten, sowie dem Task, von dem die Form derzeit bearbeitet wird, in einer Tabelle verwaltet. Eine Form-Instanz entsteht, wenn in einem Task eine neue Form generiert wird. In einer Implementierung mit aktiven Datenbanken werden die Änderungen der Prozeßstatusinformationen in diesen Tabellen zur Triggerung der Regeln verwendet.

23.3.4 Übersetzung in Datenbank-Trigger

In diesem Abschnitt wird die Realisierung des oben beschriebenen Ablaufmodells mit aktiven Datenbanken dargestellt. Analog zu dem im vorigen Abschnitt beschriebenen Ausführungsmodell gibt es für jeden der Schritte eine Gruppe von Regeln, die aus der Workflow-Beschreibung generiert wird. Für jeden Flow wird eine Regel generiert, die den „Transport“ der Forms von einem Task zum nächsten durchführt. Diese Regel triggert nach erfolgreicher Prüfung der Postcondition eines Tasks. Dies entspricht dem dritten Schritt in obigem Ausführungsmodell. Die folgende Regel spezifiziert einen Flow einer Form des Typs *form_i* von *task_A* nach *task_B*, wobei die Form gesendet wird, wenn die Bedingung *flow-condition* erfüllt ist.

```
create trigger flow_n_step3
after update of status on form_i
when new.status='finished'
and new.type=form_j and new.task = task_A and flow-condition
update new set task = task_B;
```

Die Regel feuert, wenn sich das Statusfeld in der Tabelle *form_i* verändert. Die Bedingung ist erfüllt, wenn der neue Wert des status 'finished' ist; in diesem Fall wird das task-Feld der Form auf den Nachfolgetask gesetzt.

Sämtliche Regeln können wie diese aus fixen Schablonen aufgebaut werden, das Füllen übernimmt ein Compiler, der die Regeln aus der Definition des Workflows erzeugt.

De folgenden Regeln werden aus den einzelnen Taskbeschreibungen erzeugt:

post-task rule: Diese Regel triggert, wenn der Task abgeschlossen ist, und führt die after-procedure aus (Schritt 1 des Ausführungsmodells).

postcondition rule: Diese Regel testet die postconditions (Schritt 2).

precondition rule: Diese Regel prüft die Vorbedingungen für einen Task. Dies ist notwendig, wenn der Task mehrere Eingangsflows besitzt. Bei jedem Eintreffen einer Form wird diese Regel getriggert und überprüft, ob bereits alle Voraussetzungen zur Bearbeitung des Tasks erfüllt sind (Schritt 4).

dispatch rule: Diese Regel existiert, wenn für den Task nicht ein konkreter Benutzer, sondern eine Rolle mit einem Auswahlkriterium spezifiziert wurde (Schritt 5).

Wenn das user-Feld des Tasks ausgefüllt wird, triggert die nächste Regel:

pre-task rule: Diese Regel führt die before-procedure aus. Nach Beendigung dieser Regel wird ein Signal an das Client-Programm (entweder das Benutzerinterface oder ein den Task ausführendes Programm) geschickt.

Hervorzuheben ist, daß der Workflowmanager nur aus all den Regeln, die aus der Prozeßbeschreibung generiert wurden, besteht. Alle anderen notwendigen Funktionen stellt die Datenbank zur Verfügung.

Mit konventionellen Programmen - d.h. ohne Verwendung von aktiven Datenbanken - lassen sich diese Funktionalitäten ebenfalls implementieren, es stehen dabei grundsätzlich zwei Möglichkeiten mit jeweils bedeutenden Nachteilen zur Auswahl:

1. Ein spezielles Anwendungsprogramm stellt regelmäßig Anfragen an die Datenbank (engl. polling), registriert Veränderungen und reagiert dementsprechend. Dieses Anwendungsprogramm kann ein konventionelles Programm sein oder auch ein Regelinterpreter, der an die Datenbank gekoppelt ist.

   Dieser Ansatz ist aus mehreren Gründen unzureichend: Wenn die Abfragefrequenz zu gering ist, dann könnte damit die Anwortzeit für manche Applikationen zu hoch sein, da Veränderungen in der Datenbank zu spät bemerkt werden bzw. sogar Änderungen verloren gehen. Ist die Abfragefrequenz zu hoch, wird der Rechner ständig - auch wenn kein relevantes Ereignis vorliegt - unnötig stark belastet. Frühere Arbeiten [EKW86] haben gezeigt, daß ein zentraler Scheduler in der Regel Performance-Gewinne gegenüber Sende- und Pollstrategien hat.

2. Die zweite Möglichkeit besteht darin, die Anwendungsprogramme, die Datenbankänderungen durchführen, so zu erweitern, daß diese Programme bei bestimmten Situationsveränderungen die entsprechenden Aktionen ausführen.

   Dieser Ansatz ist unzureichend, da gleiche Funktionalität auf mehrere Programme verteilt wird, jede Veränderung dieser Funktionalitäten müßte in allen Applikationen nachgeführt werden. Dies führt zu einem hohen Wartungsaufwand und erschwert die Integration bestehender Applikationen.

Die Client-Programme, die die Tasks ausführen, kommunizieren bei der Verwendung von aktiven Datenbanken direkt mit dem Datenbankserver, somit ist kein regelmäßig die Datenbank abfragender Prozeß notwendig. Anhand zweier Beispiele soll exemplarisch die Mächtigkeit des Ansatzes für die Konstruktion nicht-konventioneller Workflows dargestellt werden:

- Die Speicherung von Forms und Statusinformationen in einer Datenbank ermöglicht auch das Heranziehen von Information aus den Forms zur Kontrollbeeinflußung. Dies wird in unserem System vor allem zur Benutzerauswahl verwendet. Der Bearbeiter eines Dokuments kann in einem Eingabefeld einen Benutzer oder eine Rolle spezifizieren, an den das Dokument weitergeleitet wird. Somit kann ein einfacher Workflow generiert werden, der wie E-Mail arbeitet. Der Empfänger der Nachricht (der Form) wird einfach in der Form selbst spezifiziert (siehe Abbildung 23.4).

- Durch die Nutzung der Datenbank als aktive Komponente der Workflow-Steuerung können Änderungen, die andere (bereits bestehende) Applikationen auf den Daten der Datenbank ausführen, eine Aktivität oder einen Workflow anstoßen. Zum Beispiel könnte die Regel in Beispiel

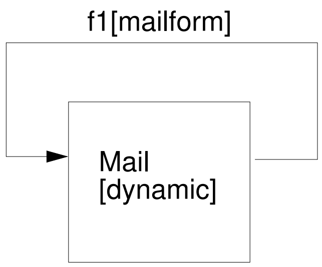

Abbildung 23.4: **Workflow zur Modellierung von E-Mail**

1, die aktiv wird, wenn die Anzahl einer Ware unter einen Schwellwert fällt, einen Bestell-Workflow auslösen.

23.4 Architektur des Prototyps - das System Panta Rhei

Zur Evaluierung unseres Ansatzes wurde ein Prototyp unter Verwendung des Datenbanksystems ORACLE, welches ein einfaches Regelsystem enthält, implementiert. Als Hardware Plattform dient ein SUN Workstation Cluster. Das System besteht aus den folgenden Komponenten:

- Server,
- Benutzerinterface-Client,
- Workflow-Designwerkzeug und WDL-Compiler und
- Monitoring und Simulation Client.

Der Server ist die aktive Datenbank mit den Regeln, die aus den Workflows erzeugt wurden. Kein zusätzlicher Code oder Prozeß ist notwendig, da die Kommunikation der anderen Komponenten ausschließlich über die Datenbank läuft.

23.4.1 Benutzerinterface-Client

Diese Komponente ist das Interface für den Benutzer des Workflowsystems, Abb. 23.5 stellt den Aufbau dar. Der typische Prozeß der Bearbeitung ist ähnlich der Bearbeitung von E-Mail: Wenn ein Task zur Ausführung bereit ist, erscheint im Benutzerinterface eine neue Zeile in der Task-Liste. Wird ein Task in dieser Liste angeklickt, sieht man rechts die Liste der zugehörigen Forms, die nun bearbeitet werden können. Dabei können nur die Felder der Forms gelesen und bearbeitet werden, bei denen dies für den jeweiligen Task definiert wurde. Wenn die Bearbeitung der Forms beendet ist, werden die

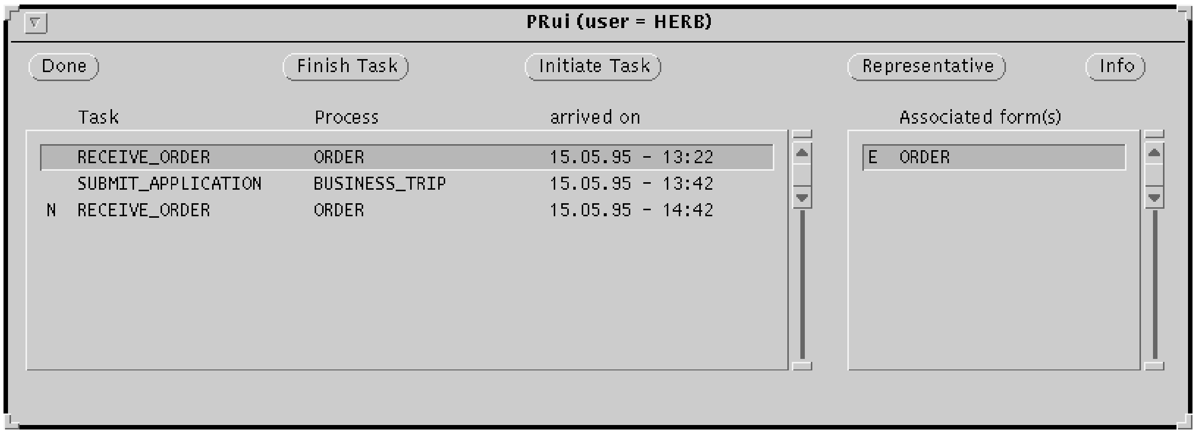

Abbildung 23.5: **Benutzerinterface**

geänderten Daten in die Datenbank übernommen. Dort wird nun die post-procedure ausgeführt und die post-condition überprüft, danach wird der Task von der Tasklist entfernt.

23.4.2 Designeditor

Zur Definition von Workflows wurde ein eigenes Designer-Interface entwickelt, das die interaktive graphische Erstellung von Workflows, Tasks und Forms ermöglicht. Das Benutzerinterface für den Workflow-Designer ist in Abb. 23.6 dargestellt.

Der zweite Teil dieses Tools ist der Workflow-Compiler, der die WDL Beschreibungen in Regeln für die aktive Datenbank übersetzt. Nachdem ein neu definierter Workflow übersetzt ist, kann der Benutzer, der den Anfangstask ausführt, den Workflow starten.

23.4.3 Monitoring und Simulation Client

Eine wichtige Aufgabe bei Workflowsystemen ist das Monitoring, d. h. die Kontrolle, welche Workflows in welchen Zuständen sind, wo Formulare „hängengeblieben“ sind oder wo Timeouts aufgetreten sind. Für diese Aufgaben wurde ein Monitoring-Client implementiert, der im Aufbau sehr ähnlich dem Interface für den normalen Benutzer ist. Der Hauptunterschied ist, daß hier alle Tasks und Forms, die gerade im System sind, sichtbar sind. Weiters ist es möglich, die Ansicht auf bestimmte oder einen bestimmten Benutzer zu fokusieren. Die Formulare sind editierbar.

Eine zweite Aufgabe dieser Komponente ist der Test und die - manuelle - Simulation von neu generierten Workflows. Man kann damit die Abläufe

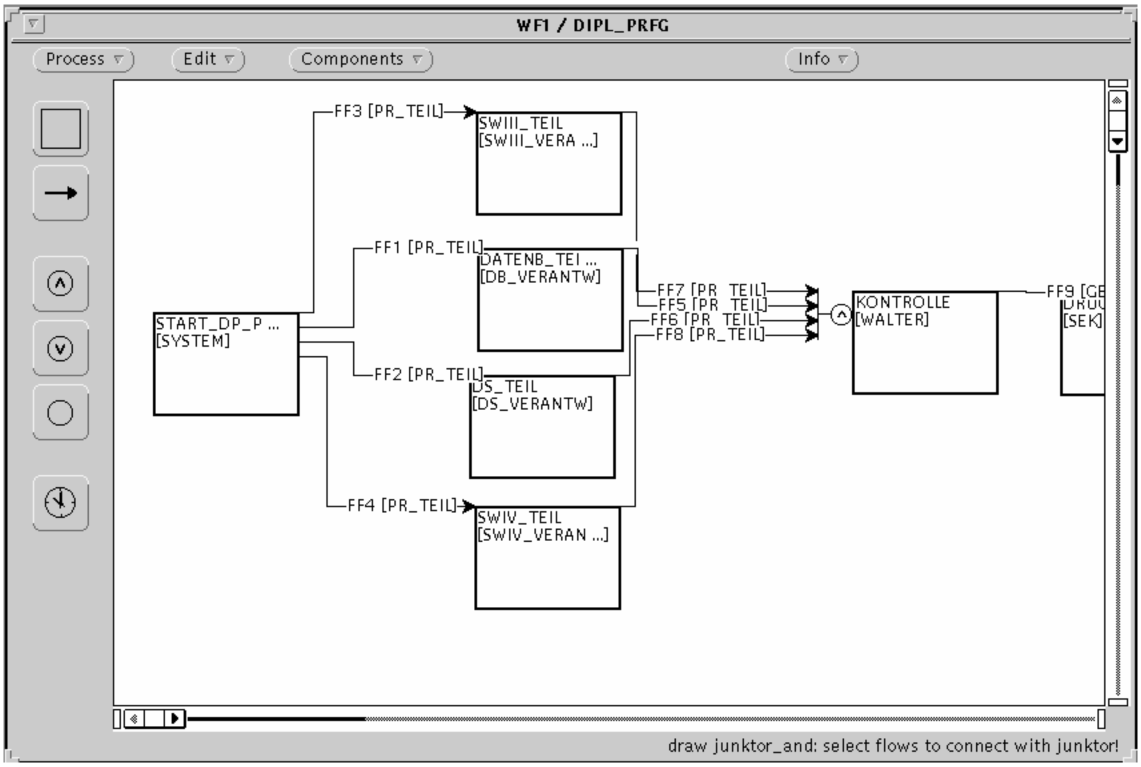

Abbildung 23.6: **Designerinterface**

in verschiedenen Variationen durchspielen und Fehler erkennen.

Die Implementierung dieser Komponente war sehr einfach, da alle notwendigen Informationen in der Datenbank vorhanden sind und die verschiedenen Views, z.B. Stati der Tasks, Belastung der Benutzer, einfach durch SQL-Abfragen zustande kommen.

23.5 Diskussion und Zusammenfassung

Wir haben eine Architektur für Workflowsysteme vorgestellt, die aktive Datenbanksysteme für die Realisierung von Workflow-Managementsystemen nutzt. Diese Architektur wurde für die Implementierung von Panta Rhei, einem prototypischen Workflow-Managementsystem, verwendet. Dieser Ansatz zeichnet sich durch eine integrierte Verwaltung von Metadaten und Instanzdaten sowie der dynamischen Prozeßdurchführung in einem System aus. Durch weitestgehende Nutzung der vom Basisystem aktive Datenbank zur Verfügung gestellten Funktionalität konnte der Prototyp mit vergleichsweise geringem Aufwand implementiert werden. Beispielsweise wird die Wiederherstellung eines konsistenten Zustand nach Fehlern direkt von der Recovery-Komponente des Datenbankssystems durchgeführt.

Die Verwendung einer aktiven Datenbank erlaubt auch eine Realisierung, bei der die Prozeßsteuerung durchgehend vorwärtsgetrieben ist, das heißt, daß im System Panta Rhei kein Prozeß durch aktives regelmäßiges Nachfragen (polling) gesteuert wird. Jede Beendigung eines Prozesses oder Tasks ändert den aufgezeichneten Pozeßstatus in der Datenbank. Trigger werden durch dieses Ereignis ausgelöst und starten die entsprechenden Folgeprozesse, falls deren Vorbedingungen alle erfüllt sind. Damit konnten die bekannten Nachteile von aktiv wartenden Prozessen vermieden werden. Es können keine Statusänderungen durch zu geringe Nachfrageraten "verloren"gehen. Das System wird auch nicht durch oftmaliges erfolgloses Nachfragen unnötigerweise belastet.

Die hier beschriebene Architektur bietet auch ein sehr hohes Maß an Flexibilität. Einerseits können Workflows sehr leicht ergänzt oder verändert werden, andererseits kann das Workflowsystem sehr einfach mit anderen Applikationen verbunden werden. Das Workflowsystem kann dabei als Integrationsplattform für andere bisher nicht integrierte Applikationen dienen.

Literaturverzeichnis

[Cha92] S. Chakravarthy, editor. *Data Engineering Bulletin, Special Issue on Active Databases*, volume 15. December 1992.

[Day88] Umeshar Dayal. Active Database Management Systems. *Proc. of the Third International Conference on Data and Knowledge Engineering, Jerusalem*, 1988.

[EB82] C. A. Ellis and M. Bernal. Office-Talk-D: An Experimental Office Information System. In *Proc. First ACM-SIGOIS Conference on Office Information Systems*, pages 131–140, 1982.

[EGN94] J. Eder, H. Groiss, and H. Nekvasil. A Workflow System Based on Active Databases. In G. Chroust, editor, *Connectivity 94*, pages 249–265. Oldenburg Verlag, 1994.

[EKW86] J. Eder, G. Kappel, and H. Werthner. Evaluation of Scheduling Mechanisms of a Dynamic Data Model by Simulation. In *Prof. of Int. Conference on Measurement and Control*, pages 86–91, 1986.

[EL95] Johann Eder and Walter Liebhart. The Workflow Activity Model WAMO. In *Proc. of the 3rd International Conference on Cooperative Information Systems*, pages 87–98, Vienna, Austria, 1995.

[GE93] Herbert Groiss and Johann Eder. Einsatz von Aktiven Datenbanken für CIM. In *Tagungsband des ADV-Kongresses Informations- und Kommunikationstechnologie für das neue Europa*, pages 861–870, 1993.

[HEA90] Heikki Hämmäinen, Eero Eloranta, and Jari Alasuvanto. Distributed Form Management. *ACM Transactions on Information Systems*, 8(1):50–76, January 1990.

[IA93] ISO and ANSI. Working Draft Database Language SQL (SQL3). Digital Equipment Corp. Maynard, MA, August 1993.

[LCS82] V.Y. Lum, D.M. Choy, and N.C. Shu. OPAS: An Office Procedure Automation System. *IBM Systems Journal*, 21(3), 1982.

[ML91] Clarence Martens and Frederick H. Lochovsky. OASIS: A Programming Environment for Implementing Distributed Organizational Support Systems. *SIGOIS Bulletin*, 12(2):29–42, 1991.

[TLF88] Michel Tueni, Jianzhong Li, and Pascal Fares. AMS: A Knowledge-based Approach to Task Representation, Organization and Coordiantion. *SIGOIS Bulletin*, 9(2):78–87, April 1988.

[WMC94b] Workflow Management Coalition. Glossary - A Workflow Management Coalition Specification. Brüssel, Belgien, 1994.

[Zlo82] M.M. Zloof. Office-By-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail. *IBM Systems Journal*, 21(3), 1982.